UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Uma abordagem para o Problema do Escalonamento Dinâmico em Projeto de Software

Trabalho de Conclusão de Curso

José Joaquim de Andrade Neto

São Cristóvão – Sergipe

2018

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

José Joaquim de Andrade Neto

Uma abordagem para o Problema do Escalonamento Dinâmico em Projeto de Software

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Leila Maciel de Almeida Silva
Coorientador(a): André Britto Carvalho

São Cristóvão – Sergipe

2018

Dedico esse trabalho a todos que me acompanharam nessa jornada
da graduação, que tanto me aconselharam e me deram suporte até o final.

Agradecimentos

Acho que cada pessoa acaba influenciando na vida da outra por mais que ela passe despercebida. No geral, esses agradecimentos são direcionados para todos os quais eu estive em contato desde o primeiro dia que pisei em sala de aula da Universidade Federal de Sergipe, em 14 de Abril de 2014. De lá para cá, meus objetivos de vida mudaram radicalmente, apesar de que sempre mantive na cabeça de que o melhor deveria ser dado, caso eu desejasse colher os frutos mais maduros dessa trilha.

Acredito que o toque final do sucesso depende mais da pessoa do que de quem está ao seu redor. Entretanto, não acredito na distinção completa dos dois. Por isso, agradeço absolutamente a todos os professores os quais tive aula durante esses quatro anos de curso. Das matérias que menos gostei até as quais me apaixonei, todos eles me ajudaram a conhecer melhor o mundo no qual estou mergulhado e explorarei mais no futuro. Aos professores que aceitaram compor a banca que julga este trabalho, meu sincero obrigado por aceitarem o convite, pela paciência e disponibilidade que reservaram para esta tarefa. Sou grato também por conta dos autores do modelo do DSPSP que eu utilizei, Shen *et al.* (2016), terem me disponibilizado o código fonte que foi útil enquanto eu estava implementando a minha versão.

Em especial, devo agradecer aos professores que mais me aproximei durante a graduação: Prof. Leila, Prof. André, Prof. Breno e Prof. Beatriz. Professora Leila, minha orientadora de pesquisa há dois anos e orientadora deste presente trabalho, foi a minha professora de grafos (duas vezes!) e de análise de algoritmos. Apesar das minhas notas baixas, não fiquei assustado ou afastado, como muitos fazem. Sua história acadêmica, repleta de sucesso e de muita luta, me serviu de inspiração e impulsionou minha vontade de melhorar, tanto para dar satisfação a mesma quanto para provar a mim mesmo de que sou capaz. Esse trabalho foi e voltou das suas mãos umas três vezes, e eu o escrevi na mão pelo menos outras duas. Ainda que nada é perfeito, sei que dei o meu melhor, e observando tudo que escrevi desde as primeiras versões, percebo o quanto evoluí especialmente por conta da senhora.

Com o professor Breno, que foi o meu professor em algumas matérias do curso, conheci uma das áreas que mais fiquei interessado em estudar, a área da teoria da computação. Aprendi a ser mais rígido com relação a escrita matemática, além de que fui recebido com respeito e boa vontade todas as vezes que busquei aprender e brinquei com sua pessoa. Professor André, meu coorientador, e Professora Beatriz desempenharam um papel importante do início ao fim dessa jornada. Se por um lado a professora Beatriz foi a minha primeira professora de programação, o professor André foi o último que sentou comigo para olhar os códigos e resultados dos experimentos deste trabalho, corrigindo e aprimorando. Juntos, esses três professores fizeram parte da coisa mais importante para mim na faculade: a Maratona de Programação.

Seria injusto não citar os nomes das pessoas que mais influenciaram as decisões de maior pesos da minha vida (as quais envolveram a maratona): Flávio Arthur, Gilberto Alexandre dos Santos, Maurício Collares e Carlos Vinícius. Esses quatro me acompanharam, do início ao fim, seja como companheiro de time, técnico ou professor. São as quatro pessoas que considero serem as mais importantes para mim na faculdade, em função do papel que a maratona desempenhou em mim. Com chuva, sol, dia, noite, madrugada, pelo menos algum deles estava estudando comigo ou me ajudando com questões ou assuntos que eu estava descobrindo, sendo que nunca faltou boa vontade da parte deles. Vocês vão longe, e já gravaram seu nome na história da maratona no departamento de computação.

Tenho certeza de que não esquecerei do dia 25 de maio de 2016, quando Flávio me confiou para montar um time com ele para a maratona daquele ano. Aqueles seis meses que se seguiram até a final brasileira, em Belo Horizonte, mostrou o quanto nós somos capazes de alcançarmos nossos objetivos quando a oportunidade aparece. Aquele resultado me deu energia e esperança para dar mais força em 2017, dessa vez com Welerson e Adam. Mesmo que não tenhamos conseguido o resultado que tínhamos em mente, sei que cresci (e crescemos!) e que o que é nosso está guardado no futuro. Valeu cada período de férias que não tivemos, cada sábado que ficamos o dia inteiro treinando e cada domingo que tivemos de ir à universidade. De aluno, fui professor um período de tempo de vários calouros, que nem sequer tinham programado na vida, mas que também dedicaram uma parte das suas férias e das manhãs de seus sábados para me ouvir ensinar sobre programação e maratona. Foi prazeroso. Faria tudo novamente, e sempre tentando ser melhor.

Por fim, devo agradecer aos meus colegas de turma, afinal foram com eles que compartilhei minha presença em sala de aula. Costumo dizer que a minha turma não é composta somente dos alunos de 2014.1. Existe somente uma turma, composta por diversas pessoas de 2014.1, 2014.2 e 2015.1. Acredito que eles também achem isso. Estivemos em todo canto da universidade, e a nossa união é percebida de longe (inclusive no RESUN, todo dia!). Particulamente, agradeço a Antônio Bispo, Gabriel Leite, Matheus Resende, Jusley Arley, Alana Lúcia, Elias Rabelo, Matheus Schettino, Patrick Jones, Victor Carity, Paulo de Brito, Jackson Tavares, Adrian Costa, Davi Silva, Antônio Carlos, Hugo Barreto, João Manoel, Itamar Souza, Rafael Andrade, João Cruz, Marina Vivas, Felipe Pereira e tantos outros por serem pessoas maravilhosas e me terem como amigo. "CC NINGUÉM DESISTE!"

Who was born in a house full of pain
Who was trained not to spit in the fan
Who was told what to do by the man
Who was broken by trained personnel
Who was fitted with collar and chain
Who was given a pat on the back
Who was breaking away from the pack
Who was only a stranger at home
Who was ground down in the end
Who was found dead on the phone
Who was dragged down by the stone
(Pink Floyd - Dogs)

Resumo

O Problema do Escalonamento em um Projeto de Software consiste na alocação de funcionários em tarefas durante o desenvolvimento de um software. Um bom escalonamento permite não somente a redução de custos e tempo de desenvolvimento, como também a minimização dos impactos que o projeto sofre quando exposto a eventos dinâmicos do mundo real como, por exemplo, a saída de um funcionário ou a chegada de novos requisitos. Este é um problema de otimização e, por ser complexo no que se refere à determinação de soluções exatas, técnicas de busca podem ser aplicadas para encontrar boas soluções. Este trabalho investiga a adequação da aplicação da meta-heurística *Nondominated Sorting Chemical Reaction Optimization* para o problema do Escalonamento Dinâmico em Projeto de Software. Uma adaptação para o problema foi proposta e validada através de experimentos a partir de comparações com as saídas geradas pela meta-heurística *Nondominated Sorting Genetic Algorithm II*. Os resultados sugerem que as duas produziram soluções que diferem estaticamente. A análise das soluções mostra que a *Nondominated Sorting Chemical Reaction Optimization* gerou saídas melhores que a *Nondominated Sorting Genetic Algorithm II* em projetos que tinham poucos funcionários e tarefas, apesar de sua eficiência cair e tornar-se pior à medida que o projeto ficava maior.

Palavras-chave: escalonamento dinâmico, meta-heurísticas, search-based, projeto de software, otimização.

Abstract

The Software Project Scheduling Problem consists of allocating employees to tasks during a software development. A good schedule allows the project not only to reduce costs and its makespan, but to minimize the impacts that the project could suffer when exposed to dynamic events of the real world, such as the exit of an employee or even the emergence of new requisits. This is a combinatorial problem, and and as it is a complex task to find good solutions, search techniques may be applied to solve the problem. This work investigates the adequation of the application of the meta-heuristic Nondominated Sorting Chemical Reaction Optimization to the Dynamic Software Project Scheduling Problem. An adaptation to the problem was proposed and validated through experiments. The results were compared with the ones of the metaheuristis Nondominated Sorting Genetic Algorithm II. The analysis showed that the Nondominated Sorting Chemical Reaction Optimization produced better outputs than the Nondominated Sorting Genetic Algorithm II in projects with few employees and tasks, despite its efficiency decreased and became worse as the project increased.

Keywords: dynamic schedule, meta-heuristics, search-based, software project, optimization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACO	Ant Colony Optimization
CRO	Chemical Reaction Optimization
DSPSP	Dynamic Software Project Scheduling Problem
GA	Genetic Algorithm
GDA	Grafo Direcionado Acíclico
KE	Kinect Energy
NSGA	Nondominated Sorting Genetic Algorithm
NSGA-II	Nondominated Sorting Genetic Algorithm II
DM	Matriz de Dedicação
MOA	Multiobjective Algorithm
MOEA	Multiobjective Evolutionary Algorithm
NCRO	Nondominated Sorting Chemical Reaction Optimization
NFL	No Free Lunch
PE	Potential Energy
SBSE	Search-Based Software Engineering
SPSP	Software Project Scheduling Problem
TCC	Trabalho de Conclusão de Curso

Sumário

1 Introdução

A área de engenharia de software é uma área ampla da computação que lida com especificações, criação de sistemas de software, bem como gerência, planejamento e desenvolvimento de sistemas computacionais (PRESSMAN, 2014). Como muitas áreas existentes atualmente, a engenharia de software busca otimizar seus processos de produção através de modelagens abstratas e precisas que permitem aos engenheiros raciocinar sobre elas, seja especificando, projetando ou implementando alguma solução para problemas da área. No entanto, o processo de otimização em engenharia de software é uma tarefa complexa. Tendo em vista que a demanda e a complexidade dos problemas crescem, faz-se necessário a automatização dos processos e da criação de novas técnicas que deem suporte para o agente que desenvolve softwares.

A Engenharia de Software Baseada em Busca – *Search Based Software Engineering* (SBSE) – é uma subárea da Engenharia de Software, introduzida por Harman e Jones (2001), que aplica técnicas de otimização para a solução de problemas que estão relacionados à engenharia de software. Soluções de problemas complexos, obtidas a partir de algoritmos de buscas, possibilitaram o avanço de subáreas da engenharia de software que antes eram exploradas, de forma limitada, devido à falta de técnicas que possibilitassem a investigação de problemas com um grande número de variáveis (HARMAN; JONES, 2001).

Os algoritmos de buscas podem ser classificados em três diferentes categorias: os métodos exatos, os aproximativos e as heurísticas. Os métodos exatos são aqueles capazes de achar a solução ótima de um problema (ROTHLAUF, 2011), mas podem ter alta complexidade. Os aproximativos não garantem a solução ótima do problema, mas são capazes de estabelecer um limite superior sobre o quão boa são as soluções dadas como saída (CARVALHO et al., 2001). As heurísticas pertencem à classe de algoritmos que, apesar de possuírem complexidade menor em relação aos apresentados anteriormente, não são capazes de garantir a otimalidade das soluções oferecidas, nem um limite sobre a qualidade das mesmas, tendo seus resultados geralmente avaliados a partir de métricas estatísticas ou de forma empírica (COELLO; LAMONT;

VELDHUIZEN, 2007).

Apesar da não garantia da solução ótima, as heurísticas são bastante estudadas e usadas em problemas de alta complexidade. Em particular, as meta-heurísticas são algoritmos de busca derivados das heurísticas, sendo os mais usados na SBSE, não somente pela sua fácil adaptação ao problema em estudo, mas também porque nem sempre é possível modelar os problemas da engenharia de software através de equações lineares (HARMAN, 2007). A classe de meta-heurísticas mais usada na SBSE é a classe das meta-heurísticas evolucionárias (COELLO; LAMONT; VELDHUIZEN, 2007), as quais se inspiram na natureza, especialmente na biologia, para a modelagem de novos algoritmos. Entre as meta-heurísticas mais utilizadas, destacam-se os Algoritmos Genéticos (GA) (MITCHELL, 1998), Colônia de Formigas (ACO) (DORIGO, 1992) e a Busca Tabu (GLOVER, 1989).

Uma das linhas de interesse da SBSE consiste em problemas relativos à gerência de projetos, a qual engloba atividades como planejamento, composição de equipes, controle de recursos, aprovação de pontos no projeto, entre outras. Dados apresentados em Group (2015) ressaltam a importância de uma boa gerência em projetos de software, uma vez que, de 50.000 projetos de várias partes do mundo, apenas 29% obtiveram sucesso[1] ao final do desenvolvimento. Pressman (2014) atribui tal ocorrência, entre vários motivos, a má escolha de funcionários e/ou suas respectivas associações para as tarefas do projeto. De fato, Chen e Zhang (2013) citam que 40% dos projetos na China que falharam tinham fracos planos de projeto e alocação de funcionários. É nesse cenário que o problema de escalonar funcionários para a realização de tarefas em um projeto de software precisa ser estudado, uma vez que um bom escalonamento ajuda a evitar (e algumas vezes impedir) incertezas e prejuízos em um projeto de software.

O Problema do Escalonamento em Projeto de Software – *Software Project Scheduling Problem* (SPSP) (ALBA; CHICANO, 2007) prevê um conjunto de tarefas e um conjunto de funcionários, que as realizará. Cada tarefa possui um custo associado, um conjunto de habilidades requeridas, e a ordem de execução deve seguir estritamente um grafo de precedência, que diz qual tarefa deve ser executada antes de outra começar. Cada funcionário possui um salário, um conjunto de habilidades e uma dedicação máxima ao projeto, podendo executar várias tarefas ao longo do dia de trabalho.

O objetivo principal do SPSP (ALBA; CHICANO, 2007) é associar os funcionários para as tarefas de tal forma que sejam minimizadas a duração do projeto e o seu custo, bem como sejam satisfeitas todas as restrições, como sobrecarga de trabalho e falta de habilidades. O problema do escalonamento tem sido investigado, em sua maioria, com as técnicas aplicadas na SBSE (MINKU; SUDHOLT; YAO, 2014), evidenciando o uso de meta-heurísticas. Além das citadas anteriormente, há trabalhos que investigam o SPSP com algoritmos inspirados em Reações Químicas (ANDRADE; SILVA; CARVALHO, 2017), no fluxo da água que percorre um rio (CRAWFORD et al., 2016), na organização de formigas andando em uma trilha (CRAWFORD

[1] O sucesso é definido pela entrega no prazo estimado e com um custo satisfatório com o previamente orçamentado.

et al., 2014), entre outros.

No entanto, apesar do SPSP proposto em (ALBA; CHICANO, 2007) estar sendo continuamento investigado (FERRUCCI; HARMAN; SARRO, 2014), não se trata de um modelo robusto para a realidade. Isso acontece, pois, supõe-se que uma vez que os funcionários são escalonados, estes permanecerão em suas respectivas atividades até o final do projeto, com as dedicações pré-estabelecidas. Contudo, Pressman (2014) cita que vários eventos da vida real podem influenciar em um projeto de software, e que alguns destes não podem sequer ser previstos. Alguns destes eventos são: dificuldades humanas que não puderam ser previstas no momento atual, subestimação dos recursos que serão usados no projeto, mudanças nos requisitos do software, riscos não previsíveis no início do projeto, entre outros.

Assim, a dinamicidade da vida real também deve ser incluída na formulação SPSP, destacando a necessidade de incorporar eventos de incertezas e de dinâmica de projeto. Neste contexto, o SPSP foi estendido para o Problema do Escalonamento Dinâmico em Projeto de Software – *Dynamic Software Projet Scheduling Problem* (DSPSP). Shen et al. (2016) relatam que, apesar de não existirem muitos trabalhos na literatura acerca do escalonamento dinâmico, sua importância para o auxílio no gerenciamento de projetos de software não deve ser subestimada. No modelo proposto por eles, o DSPSP considera eventos dinâmicos, alguns deles relacionados à associação e desassociação de funcionários de uma dada tarefa, e até mesmo às incertezas de projeto que causam alteração no valor de certas tarefas. As soluções para o problema são encontradas pela aplicação de meta-heurísticas evolucionárias. Investigar o DSPSP é o objeto central desta monografia.

1.1 Hipótese do Trabalho

Estudos preliminares do autor com o problema do SPSP (ANDRADE; SILVA; CARVALHO, 2017) evidenciam que a meta-heurística baseada em Reações Químicas encontrou boas soluções, quando comparadas a resultados obtidos por outros algoritmos utilizados na SBSE, como o NSGA-II (DEB et al., 2002). Diante destes resultados preliminares, a hipótese deste trabalho é de que é possível aplicar a meta-heurística *Nondominated Sorting Chemical Reaction Optimization* (NCRO) (BECHIKH; CHAABANI; SAID, 2015) ao DSPSP proposto em (SHEN et al., 2016) e obter resultados equiparáveis ou melhores ao do NSGA-II.

Para fomentação da hipótese, ressalta-se a importância da investigação de diversos métodos de buscas para um dado problema, uma vez que, no caso de meta-heurísticas, é impossível propor uma que ofereça boas soluções para todos os problemas (HO; PEPYNE, 2002).

1.2 Objetivo do Trabalho

O objetivo deste trabalho é investigar a adequação do NCRO para o DSPSP, contribuindo, assim, para um melhor entendimento do tema. O objetivo é justificado pelo fato de que tanto o DSPSP quanto o NCRO foram propostos recentemente e não há trabalhos que investiguem a aplicação dessa meta-heurística para o DSPSP. Na realidade, existem poucos trabalhos na literatura que estudam o DSPSP, de modo que a investigação aqui realizada é um interessante tema de pesquisa.

Para alcançarmos o objetivo proposto, alguns objetivos específicos foram definidos, a seguir: (i) definição do modelo DSPSP a ser utilizado; (ii) implementação do modelo definido em (i); (iii) adaptação da meta-heurística; (iv) realização dos experimentos; (v) análise dos resultados.

1.3 Metodologia

A metodologia usada para a elaboração deste trabalho foi dividida em diversas fases, as quais foram compostas por diversas atividades, descritas a seguir:

- **Estudo dos conceitos** de toda a teoria, como problemas com muitos objetivos, meta-heurísticas, métricas de avaliação e o problema do escalonamento, dentre outros. Também foram estudadas as ferramentas usadas, possibilitando assim, o desenvolvimento deste trabalho.
- **Revisão sistemática** sobre o Problema do Escalonamento Dinâmico em Projeto de Software. A revisão seguiu o protocolo proposto em (KITCHENHAM, 2004) e (KITCHENHAM; CHARTERS, 2007), e está apresentada no Apêndice A.
- **Implementações e adaptações** tanto do modelo do DSPSP sugerido por Shen et al. (2016) quanto da meta-heurística NCRO para este problema.
- **Realização dos experimentos** no jMetal aplicando ao modelo do DSPSP no NCRO e no NSGA-II, através de múltiplas execuções, com o intuito de obter uma amostra confiável para a comparação dos resultados. O NSGA-II é uma meta-heurística bastante utilizada em SBSE e neste trabalho serve de referência para a comparação dos resultados obtidos.
- **Análise dos resultados** aplicando métricas de análise de soluções em problemas com muitos objetivos, como o *Hypervolume*, e testes estatísticos, como o Teste de Wilcoxon.

1.4 Organização

Este trabalho a ser apresentado nos próximos capítulos está organizado da forma descrita a seguir. O Capítulo 2 compreende a revsisão da literatura, onde os conceitos que irão embasar todos os algoritmos e métodos usados no modelo e nas meta-heurísticas são descritos. A revisão da litertura inclui os trabalhos relacionados que direcionaram a escolha do modelo e dos algoritmos usados neste trabalho.

O Capítulo 3 apresenta as métricas usadas e os passos seguidos para a realização dos experimentos, bem como a discussão da execução e análise dos resultados obtidos. As conclusões, expressas no capítulo 4, apresentam um resumo do que foi feito, tecendo considerações finais sobre o trabalho e dos resultados dos experimentos realizados, além de sugerir para trabalhos futuros. Por fim, o Apêndice A detalha a revisão sistemática feita para coletar a literatura relacionada ao DSPSP, e o Apêndice B ilustra todos os gráficos gerados pelos testes realizados nos experimentos.

2

Revisão da Literatura

Para a investigação da hipótese proposta neste trabalho, é necessário, primeiro, entender as definições formais acerca do problema do escalonamento. Como já foi mostrado, os objetivos do SPSP são achar um escalonamento para os funcionários, tal que este resulte na menor duração de produção e menor custo de desenvolvimento. Problemas desse tipo, com mais de um objetivo, recebem o nome de problemas multiobjetivos. Após a apresentação do formalismo e dos conceitos que envolvem tais problemas, é discutido o SPSP e sua versão dinâmica, objeto de estudo desse trabalho, o DSPSP, juntamente com a literatura relacionada. São apresentadas, também, as técnicas de obtenção de soluções no contexto da SBSE, através da introdução de meta-heurísticas e do detalhamento das meta-heurísticas NCRO e NSGA-II, aplicadas ao DSPSP.

2.1 Problemas com Muitos Objetivos

Um problema é dito multiobjetivo quando ele possui mais de um objetivo, k, por exemplo, a ser otimizado simultaneamente. Formalmente, uma otimização com vários objetivos é composta por vetores de decisões $x = [x_1, x_2, ..., x_n]^T$, onde n são as variáveis de decisões e T é a transposição do vetor coluna - a representação usual - para o vetor linha, por restrições de modelo – limitações físicas ou temporais, por exemplo – que são representadas por inequações $g_i \leq 0, i = \{1, 2, ..., m\}$ ou igualdades $h_j(x) = 0, j = \{1, 2, ..., p\}$ onde m e p representam a quantidade de restrições. Para a medição da qualidade de uma solução, usa-se funções objetivos, expressas a partir dos vetores de decisões. Uma função objetivo é definida como $f(x) = [f_1(x), ..., f_z(x)]^T$, onde z é o número de objetivos (COELLO; LAMONT; VELDHUIZEN, 2007).

Quando um problema possui somente um objetivo, existe apenas um valor para a solução ótima (LUNA et al., 2013). No entanto, quando o número de objetivos é maior do que 1, o conceito de uma solução boa muda, pois a noção de otimalidade sofre uma variação. Boas

soluções são aquelas que possuem um bom compromisso entre os objetivos, fazendo parte de um balanceamento entre os resultados (COELLO; LAMONT; VELDHUIZEN, 2007). A solução que possui essa característica, de um problema multiobjetivo, é dita pertencer ao Conjunto Ótimo de Pareto, indicando que um objetivo não pode ser melhorado sem que prejudique algum outro. O Conjunto Ótimo de Pareto passa a ser chamado de Fronteira de Pareto quando plotado no espaço dos objetivos (GOLDBARG; GOLDBARG; LUNA, 2016). Matematicamente, define-se que (COELLO; LAMONT; VELDHUIZEN, 2007):

Definição 2.1.1 (Ótimo de Pareto) *Uma solução x^* é ótima de Pareto se não existir nenhum $x \in X$ tal que $f_z(x) < f_z(x^*)$ para todo $z \in K = \{1,2,...,k\}$. Essa definição mostra que se x^* é ótimo de Pareto então não existe nenhum vetor x que irá melhorar um objetivo sem causar, simultaneamente, a piora de um outro.*

Definição 2.1.2 (Dominância de Pareto) *É dito que uma solução $u = (u_1, u_2, ..., u_z)$, domina uma outra solução $v = (v_1, v_2, ..., v_z)$ se e se somente se para todo $z \in K = \{1,2,...,k\}$ tem-se que $f_z(u) \leq f_z(v)$ e $\ni z : f_z(u) < f_z(v)$.*

Definição 2.1.3 (Fronteira de Pareto) *É a imagem de um determinado conjunto de soluções ótimo de Pareto no espaço dos valores objetivos.*

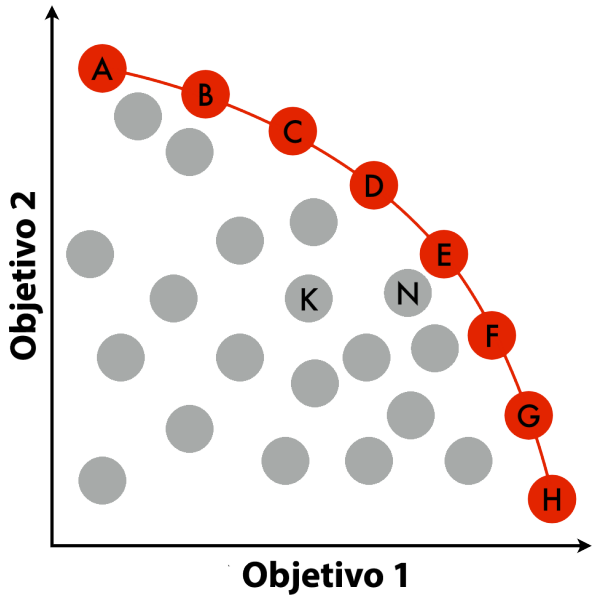
A Figura 1 exemplifica uma Fronteira de Pareto extraída de um conjunto de soluções. Ela está representada por um plano cartesiano pois está relacionada um problema com dois objetivos. Ainda, é referente a um problema de maximização. Os pontos em vermelho (A, B, ..., H) fazem parte da fronteira de Pareto. Observa-se que, concordando com a Definição 2.1.2, nenhum ponto que faz parte da fronteira domina o outro por completo, isto é, há, ao menos um objetivo no qual ele não é melhor do que outra solução pertencente ao mesmo conjunto.

2.2 O Problema do Escalonamento em Projeto de Software

O projeto de um software é composto de tarefas e funcionários, que em diferentes cenários e ambientes de trabalho, se organizam e planejam diferentes formas de fazer um produto. Independentemente da demanda e do mercado alvo, é crucial que antes do projeto haja uma organização e divisão sobre quem irá fazer o que durante o desenvolvimento do software, a fim de respeitar todos os prazos e orçamentos (SOMMERVILLE, 2006). Essa divisão é comumente chamada de fase de escalonamento, onde cada funcionário é associado a uma ou mais tarefas e é definida sua participação na mesma. Tal escalonamento deve respeitar, também, a capacidade (habilidade ou contratual) de cada funcionário.

A tarefa de apresentar um escalonamento é realizada pelo líder do projeto. Segundo Pressman (2014), o líder deve fazer um escalonamento de tal forma que seja possível para ele

Figura 1 – Fronteira de Pareto de um exemplo do SPSP.



Fonte: Adaptado de (WIKIPÉDIA, 2018).

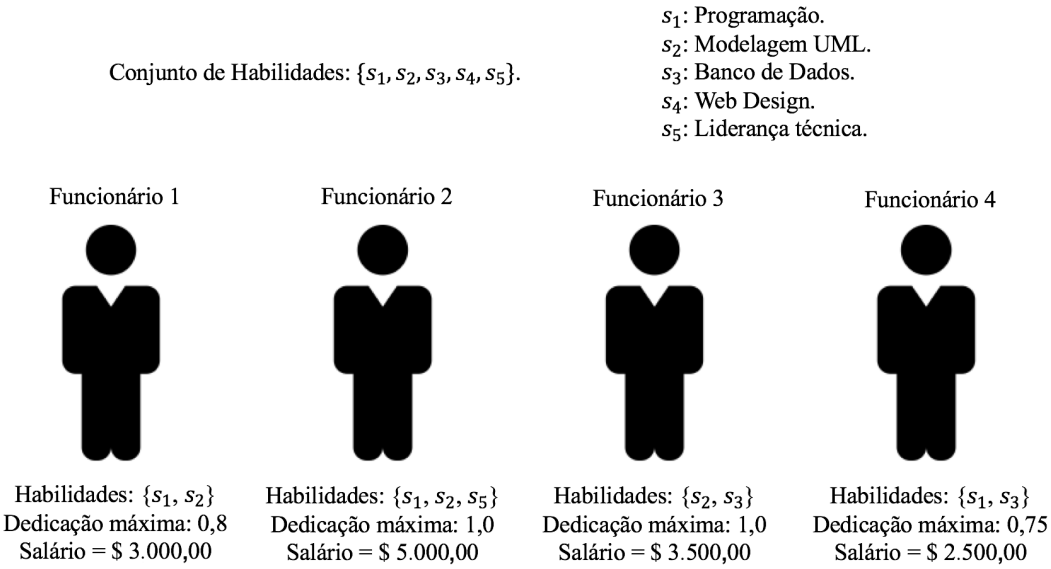
controlar e monitorar o progresso do projeto. Sem um escalonamento formalmente definido, é impossível fazer isso caso o projeto seja complexo demais (PRESSMAN, 2014). Como já mencionado, Chen e Zhang (2013) mostram que na China, por exemplo, mais de 40% dos projetos que falharam tinham fracos planos de projeto e funcionários. Já (GROUP, 2015) relatou que somente 29% de 50.000 projetos do ano de 2015, obtiveram sucesso. O relatório ainda mostra que, se avaliados por tamanho, apenas 2% dos projetos considerados grandes puderam chegar ao fim do desenvolvimento de forma satisfatória.

O problema de escalonar pessoas a tarefas em um projeto de software é estudado na literatura e é conhecido como o Problema do Escalonamento em Projeto de Software. O SPSP é uma subárea dentro da área relacionada a desenvolvimento de software, a tarefa de gestão de um projeto de software (FERRUCCI; HARMAN; SARRO, 2014), (PEIXOTO; MATEUS; RESENDE, 2014). Segundo Pressman (2014), a atividade de escalonar distribui o esforço estimado ao longo de toda a duração do desenvolvimento, alocando esse esforço em tarefas específicas do ciclo do desenvolvimento. Portanto, o SPSP consiste em associar um conjunto de funcionários a determinadas tarefas, seguindo regras e visando o maior lucro possível.

Como exemplo, considera-se as Figuras 2 e 3, onde ambas representam um possível cenário para um projeto de software. A Figura 2 elenca o conjunto de funcionários designados para o projeto. Cada funcionário possui habilidades particulares, assim como uma dedicação ao trabalho e seu salário (o funcionário 1, em particular, possui habilidades de programação e modelagem UML, dedicação de 80% e um salário de $\$3.000,00$). Para as tarefas, na Figura 3, a representação é similar, dada a partir de um Grafo Direcionado Acíclico (GDA). Cada tarefa possui um custo associado a ela, e habilidades necessárias para a sua realização. Observa-se que,

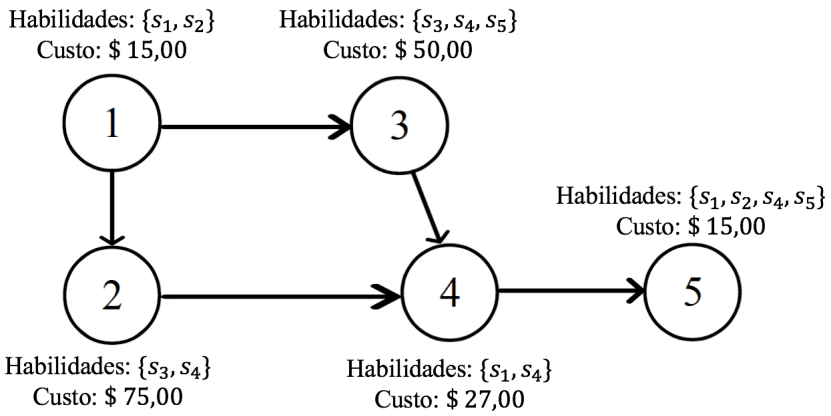
para a tarefa 1, o seu custo de produção é estimado em $15 unidades monetárias, possuindo a programação e a modelagem UML como habilidades necessárias. Nesse caso, o funcionário 1 será capaz de realizá-la.

Figura 2 – Representação de um funcionário e suas propriedades.



Fonte: Próprio autor.

Figura 3 – Representação de uma tarefa e suas propriedades.



Fonte: Próprio autor.

Uma solução para o SPSP está representada na forma de uma matriz, denominada de Matriz de Dedicação (DM). Nela, cada célula indica o quanto um funcionário irá dedicar-se para uma tarefa, em um dia de trabalho. Linhas representam o funcionário, enquanto que as colunas representam as tarefas. A Figura 4 ilustra uma matriz de dedicação. Nela, por exemplo, o funcionário 3 dedica 100% do seu dia de trabalho ao projeto, enquanto que o Funcionário 4 gasta 95% de seu dia de trabalho no projeto. Se o valor de uma célula é $0,0$, então quer dizer que o funcionário em questão não trabalha nessa tarefa. Como o objetivo de todo líder de projeto é de

que o tempo de desenvolvimento seja o menor possível, gastando-se uma quantidade mínima de recursos (ALBA; CHICANO, 2007), o escalonamento deve ser feito de forma ótima, observando inclusive a várias restrições, que pode ser falta de habilidade por parte dos funcionários ou até mesmo excesso de carga de trabalho.

Figura 4 – Matriz de dedicação de um escalonamento.

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5
Funcionário 1	0,00	0,32	0,32	0,10	0,06
Funcionário 2	0,10	0,10	0,10	0,40	0,30
Funcionário 3	0,25	0,25	0,25	0,15	0,10
Funcionário 4	0,75	0,05	0,05	0,05	0,05

Fonte: Próprio autor.

Durante o processo do escalonamento, erros podem acontecer, podendo ser de diferente natureza (humana ou tecnológica, por exemplo). Tais erros podem conduzir à construção de uma matriz de dedicação que viola uma ou várias das restrições impostas pelo modelo. Na DM da Figura 5, o Funcionário 1 excede sua carga de trabalho, uma vez que sua dedicação máxima é de 80% do seu dia de trabalho, mas foram escalonados para ele uma dedicação total de 100%. Uma tentativa de resolução desse problema seria a realização de um novo escalonamento. Porém, uma vez computada, a matriz de dedicação não é alterada no modelo do SPSP.

Figura 5 – Matriz de dedicação gerada violando a restrição de dedicação (funcionário 1).

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5
Funcionário 1	0,00	0,30	0,30	0,30	0,10
Funcionário 2	0,10	0,10	0,10	0,40	0,30
Funcionário 3	0,25	0,25	0,25	0,15	0,10
Funcionário 4	0,75	0,05	0,05	0,05	0,05

Fonte: Próprio autor.

A impossibilidade de um novo escalonamento no modelo apresentado em (ALBA; CHICANO, 2007) fez com que novos estudos e modelos robustos surgissem, observando-se que um modelo real do problema do escalonamento deve se capaz de trabalhar com todas essas condições e ao mesmo tempo oferecer um escalonamento que seja humanamente executável. Tal variante é conhecida na literatura como o Problema do Escalonamento Dinâmico em Projeto de Software (SHEN et al., 2016).

2.2.1 O Problema do Escalonamento Dinâmico em Projeto de Software

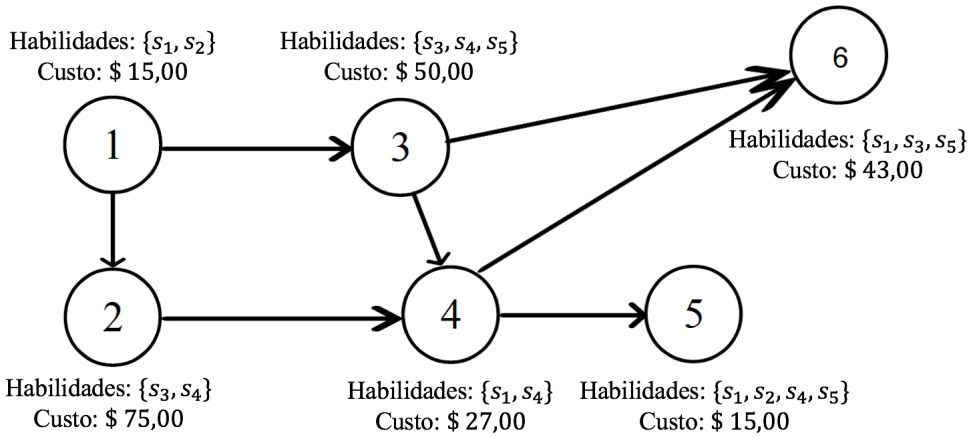
O Problema do Escalonamento Dinâmico em Projeto de Software consiste em formar um escalonamento e refazê-lo, caso necessário, à medida que o projeto sofre ações resultantes de

eventos incertos, como mencionado anteriormente. Nessa nova modelagem, é assumido que em qualquer ponto do desenvolvimento do projeto a matriz de dedicação pode ser alterada (SHEN et al., 2016). A chegada ou remoção de tarefas e/ou funcionários são exemplos de acontecimentos que influenciam na escolha do líder do projeto de refazer o escalonamento, a fim de manter os objetivos de terminar o projeto com o menor custo e a menor duração possível.

É possível encontrar na literatura três diferentes maneiras de tratar o escalonamento dinâmico. São elas: completamente reativo, preditivo-reativo e o escalonamento proativo (OUELHADJ; PETROVIC, 2008). Um escalonamento completamente reativo cria escalonamentos parciais para algum futuro imediato, baseado nas informações atuais. Um escalonamento preditivo-reativo realiza um novo escalonamento a partir de anteriores, adaptando-o, assim, para o estado atual. Já um escalonamento proativo procura gerar escalonamentos antecipadamente, podendo satisfazer requisitos previsíveis em um ambiente dinâmico (OUELHADJ; PETROVIC, 2008).

A alteração do conjunto de tarefas implica a necessidade da reavaliação do escalonamento. Ao adicionar uma nova tarefa, a tarefa 6, ao conjunto de tarefas citado anteriormente, na Figura 8, o GDA resultante pode ser o da Figura 6, por exemplo. Isso obriga o reescalonamento dos funcionários, gerando uma nova matriz de dedicação, similar à da Figura 7. Para que sejam minimizados os prejuízos que podem ser ocasionados com a mudança na DM, um modelo dinâmico do SPSP também deve considerar a estabilidade, em relação a mudanças abruptas, e à robustez, diante da variância que o custo das tarefas podem assumir, características estas presentes em eventos dinâmicos.

Figura 6 – Associação da tarefa 6 ao projeto.



Fonte: Próprio autor.

2.2.2 Trabalhos Relacionados

Apesar da relevância do problema do escalonamento para a engenharia de software, a qual é ressaltada em relatórios científicos, como Group (2015), poucos trabalhos na literatura estudam

Figura 7 – DM criada a partir do reescalonamento devido a chegada de uma nova tarefa.

	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
Funcionário 1	0,00	1,0	0,34	0,20	0,40	0,00
Funcionário 2	0,10	0,10	0,10	0,40	0,30	0,10
Funcionário 3	0,25	0,25	0,25	0,15	0,10	0,10
Funcionário 4	0,75	0,05	0,05	0,05	0,05	0,25

Fonte: Próprio autor.

o Problema do Escalonamento em sua forma dinâmica. Ainda, os modelos existentes na literatura que procuram ser mais próximos à realidade são mais custosos, em termos computacionais. Por outro lado, os que optam por uma modelagem mais simples, acabam restringindo os seus domínios, deixando de lado características, referentes ao tema, que tornariam o modelo mais real.

Dentre os trabalhos que abordam o DSPSP, Gueorguiev, Harman e Antoniol (2009) aplicaram meta-heurísticas evolucionárias para encontrar a fronteira de Pareto, onde as soluções representam um compromisso entre os objetivos duração e robustez, esta última definida como a diferença entre o tempo de duração à medida que novas tarefas chegavam ou tinham seus tempos de duração acrescidos.

Antoniol, Penta e Harman (2004) usaram um algoritmo genético para encontrar a melhor ordem de processamento das tarefas e a melhor alocação dos funcionários para os times do projeto. Uma fila foi usada para analisar a sensibilidade dos resultados obtidos pelo algoritmo genético, em relação às incertezas causadas por conta das imprecisões no cálculo dos preços das tarefas, por abandono e até mesmo re-execução das mesmas. Essa análise permitiu guiar o processo de busca do algoritmo. No entanto, esse processo pode ter que ser feito várias vezes até que seja apresentada uma solução satisfatória, o que pode ser custoso. Chicano et al. (2012) propuseram uma nova abordagem multiobjetiva ao SPSP, a qual considera a produtividade dos funcionários, quando estes trabalham em diferentes tarefas, e nas imprecisões dos preços das tarefas. É assumido que a variância dos preços seguem uma distribuição uniforme, e a robustez do projeto é dada a partir do desvio-padrão da duração do projeto e do valor obtido a partir de um certo número de simulações dos preços das tarefas.

Xiao et al. (2010) consideraram o re-escalonamento dos funcionários associando eventos dinâmicos que ocorrem ao longo do projeto. No entanto, diferentemente dos trabalhos já citados, há algumas limitações nesse modelo. Primeiro, a associação de um funcionário a uma tarefa não é determinado pela meta-heurística, e sim na entrada do algoritmo. Segundo, apenas a chegada de uma tarefa é considerada, enquanto que no mundo real vários eventos dinâmicos ocorrem. Terceiro, apesar de possuir vários objetivos, a estabilidade do projeto, em relação aos eventos dinâmicos, e a utilidade (importância para resposta diante dos eventos dinâmicos) são combinadas em um único valor através de uma soma ponderada. Uma vez que os objetivos são

conflitantes entre si, uma abordagem multiobjetiva, que permite a geração de uma fonteira de Pareto, pode ser mais vantajosa.

Por fim, Shen et al. (2016) estendem os trabalhos citados anteriormente com uma proposta multiobjetiva, onde quatro tipos de eventos dinâmicos podem acontecer em qualquer momento do desenvolvimento do projeto. São eles: funcionário sai e retorna a tarefa, novas tarefas surgem e imprecisões nos preços das mesmas. A robustez do projeto, em relação a imprecisão dos preços das tarefas, e o reescalonamento imediato, diante dos eventos dinâmicos são inspirados nos métodos proativos. A alocação e a dedicação dos funcionários nas tarefas são definidas de forma dinâmica, uma vez que essa estratégia é mais realística. O modelo proposto possui quatro objetivos distintos, duração, custo, robustez e estabilidade, que são otimizados simultaneamente. Por incorporar essa abordagem multiobjetiva, que permite o uso de técnicas de buscas, como meta-heurísticas, este trabalho irá investigar o DSPSP sob a ótica do modelo de Shen et al. (2016).

2.3 Representação Formal do DSPSP

Essa seção irá detalhar os principais elementos do DSPSP, necessários para o entendimento do modelo.

2.3.1 Eventos Dinâmicos no Problema do Escalonamento

No DSPSP eventos dinâmicos e incertos podem ocorrer a qualquer instante em um projeto de software. Para simular tais eventos, é necessário defini-los formalmente, de tal forma que seja possível computar suas consequências no escalonamento do cenário atual. Seguindo o modelo proposto em (SHEN et al., 2016), três eventos dinâmicos e um de incerteza são incorporados ao modelo do SPSP. São eles:

Evento 1 - Funcionário sai da tarefa: vários são os motivos para um funcionário sair de uma tarefa, entre eles, ficar doente, participar de vários projetos ao mesmo tempo, aprender novas habilidades, etc.

Evento 2 - Funcionário retorna a tarefa: um funcionário pode voltar a uma tarefa em qualquer instante depois que ele saiu dela. A diferença entre o tempo que ele voltou com o instante em que ele saiu é denominada de "*funcionário voltou*".

Evento 3 - Novas tarefas surgem: novas tarefas podem surgir à medida que o projeto vai sendo executado. Alterações nos requerimentos do software ou decisões tomadas pelo *stakeholders* podem ser a causa para a adição de novas tarefas. Uma nova tarefa pode ser considerada urgente ou regular. Tarefas urgentes são aquelas que necessitam execução imediata, enquanto que tarefas regulares são executadas durante o desenvolvimento do projeto.

Evento 4 - Preços incertos em tarefas: modificações nas especificações de uma tarefa e imprecisão no início do projeto podem fazer com que os preços da tarefa variem. As variações seguem uma distribuição normal $\phi(\mu,\sigma)$. Em cada tarefa são atribuídos diferentes valores de média de custo, μ, e de desvio padrão, σ, sendo que o valor médio é escolhido como o custo inicial de uma tarefa.

2.3.2 Propriedades dos funcionários

Funcionários são recursos importantes para o DSPSP, visto que o custo de um projeto está fortemente ligado à dedicação dos mesmos ao projeto. Supondo M o conjunto de funcionários, têm-se que, para um dado projeto, há $|M|$ funcionários a serem alocados nas tarefas, onde S é o conjunto de habilidades necessárias para a execução das mesmas. A Tabela 1 sumariza todas as variáveis relativas a um funcionário no DSPSP.

Seja $t_l = (0,1,2,...)$ um instante qualquer a partir do momento que um método de reescalonamento é acionado. Usaremos a notação $x(t_l)$ para simbolizar que a variável x é relativa ao instante de tempo t_l. As propriedades dos funcionários consideradas nesse modelo não sofrem alteração à medida que o projeto é executado, com exceção do atributo $e_i^{disponivel}(t_l)$. Isso acontece, pois, pode ocorrer a situação de que um funcionário e_i tenha que sair do projeto, e talvez volte. Assim sendo, o atributo $e_i^{disponivel}(t_l)$ é representado por uma variável binária, onde 1 significa que o funcionário e_i está disponível no tempo t_l, e 0, caso contrário. Define-se o conjunto $e\_disp(t_l)$ como sendo o conjunto de todos os funcionários disponíveis no instante t_l, ou seja, $e\_disp(t_l) = \{e_i | e_i^{disponivel}(t_l) = 1, \forall\ i = (1,2,...,|M|)\}$.

A variável e_i^{hab} representa o conjunto, de tamanho $|S|$, que quantifica a proficiência do funcionário e_i nas habilidades do projeto. Dessa forma, $e_i^{hab} = \{pro_i^1, pro_i^2, ..., pro_i^{|S|}\}, pro_i^k \in [0,C]$ são os indicadores que dizem o quão bom o funcionário e_i é na habilidade pro^k. O valor de C é definido como parâmetro de entrada. Caso $pro_i^k = 0$, então e_i não domina a habilidade k. Da mesma forma, se $pro_i^k = C$, então e_i domina totalmente a habilidade k. Portanto, a variável hab_i é definida como sendo $hab_i = \{k \mid pro_i^k > 0, k = 1,2,...,|S|\}$.

A variável $e_{ij}^{proficiencia}$ relaciona de forma direta os funcionário às tarefas. Ela quantifica a proficiência do funcionário e_i na tarefa $\mathscr{T}_j$, e não sofre alteração durante a execução do projeto. Seu valor é calculado como sendo $e_{ij}^{proficiencia} = \prod_{k \in req_j} \frac{pro_j^k}{C}$, onde a símbolo $\prod$ indica o produto de valores, e req_j o conjunto de habilidades necessárias para a execução da tarefa $\mathscr{T}_j$. Nota-se, ainda, que $e_{ij}^{proficiencia} \in [0,1]$.

2.3.3 Propriedades das tarefas

As tarefas também são importantes para o DSPSP pois são elas que guiam o fluxo do projeto. Existem tarefas que necessitam que tarefas anteriores a ela sejam executadas para que a mesma possa começar. Essa característica permite modelar a relação entre as tarefas a partir de

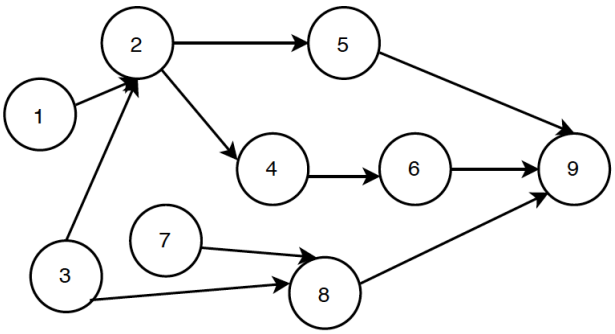
Tabela 1 – Variáveis de um funcionário no DSPSP.

Tipo	Descrição
e_i	Um funcionário no projeto.
e_i^{hab}	Indicadores de quanto o funcionário e_i domina as habilidades do projeto.
hab_i	Conjunto de habilidades que o funcionário e_i possui alguma proficiência.
e_i^{maxded}	Dedicação máxima do funcionário e_i ao projeto.
$e_i^{salario}$	Salário normal do funcionário e_i.
$e_i^{exc\_salario}$	Salário do funcionário e_i quando e_i^{maxded} é excedido.
$e_i^{disponivel}(t_l)$	Booleano que indica se o funcionário e_i está disponível em t_l.
$e_{ij}^{proficiencia}$	Proficiência do funcionário e_i na tarefa $\mathscr{T}_j$

Fonte: Próprio autor.

um Grafo Direcionado Acíclico $G(V,E)$, e portanto, ao menos uma tarefa não pode possuir um predecessor, de tal forma que esta será a primeira tarefa a ser executada. A Figura 8 ilustra um possível GDA. O conjunto de vértices, V, é representado pelas tarefas, enquanto que o conjunto das arestas, E, indica a relação de precedência entre as tarefas. Seja uma aresta $(\mathscr{T}_i, \mathscr{T}_j) \in E$, então isso significa que a tarefa $\mathscr{T}_j$ só pode ser iniciada após o final da tarefa $\mathscr{T}_i$. Se a tarefa $\mathscr{T}_j$ possuir vários antecessores, então ela será iniciada assim que a última tarefa antecessora seja finalizada.

Figura 8 – Grafo Direcionado Acíclico representado para o DSPSP.



Fonte: Próprio autor.

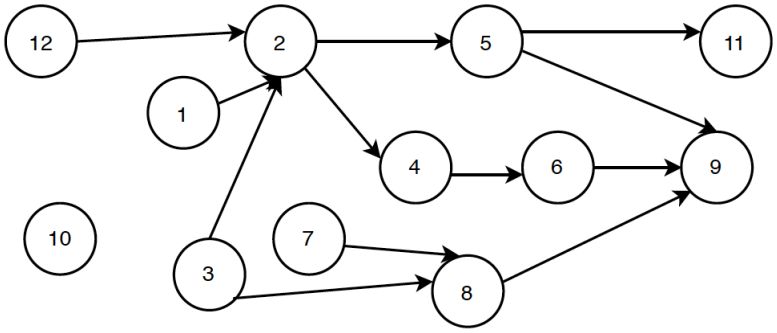
Suponha que no instante inicial do projeto, t_0, há $N(t_0)$ tarefas. De acordo com o **Evento 3**, novas tarefas podem chegar, de tal forma que, em um instante t_l, o número de tarefas pode ser acrescido com mais $N^{'}(t_l)$ tarefas. Dessa forma, o número de tarefas no instante t_l passa a ser $N(t_l) = N(t_0) + N^{'}(t_l)$. A Tabela 2 sumariza todas as variáveis relativas a uma tarefa no DSPSP. Cada Tarefa $\mathscr{T}_j$ possui características que independem do instante atual do projeto, sendo elas o custo estimado ($\mathscr{T}_j^{c\_est\_t}$), o conjunto de habilidades necessárias para a realização sua realização (req_j) e a respectiva distribuição normal $\phi(\mu_j, \sigma_j)$, relativas ao **Evento 4**.

Algumas variáveis relacionadas às tarefas podem sofrer alterações à medida que o tempo avança. Por exemplo, o GDA está sujeito à adição e/ou retirada de tarefas, de acordo com com o

Evento 3, relacionado a dinâmica em projeto de software, mencionado anteriormente. Quando uma tarefa chega, sua prioridade é verificada, e esta pode ser adicionada como um vértice sem precedentes (caso seja urgente), ou dependente de uma tarefa ainda não realizada. A Figura 9 ilustra um caso: com a chegada das tarefas 10, 11 e 12, suas prioridades são verificadas. Com menor prioridade, a tarefa 11 é posta no final do projeto, enquanto que a 10 e a 12, por possuírem urgência, são inseridas sem precedentes.

Além das alterações no GDA, as variáveis $\mathscr{T}_j^{acabou}(t_l)$ e $\mathscr{T}_j^{disponivel}(t_l)$ também podem ser alteradas em qualquer instante do projeto. Isso acontece porque as tarefas só podem ser executadas se houverem funcionários associados a elas que possuam habilidades necessárias para executá-las e todas as suas precedências já estiverem sido executadas. Dessa forma, define-se $\mathscr{T}\_disp(t_l)$ o conjunto de tarefas disponíveis no instante t_l, isto é, aquelas em que $\mathscr{T}_j^{disponivel}(t_l) = 1$.

Figura 9 – Adição das tarefas 10, 11 e 12.



Fonte: Próprio autor.

Tabela 2 – Variáveis de uma tarefa no DSPSP.

Tipo	Descrição
$\mathscr{T}_j$	Uma tarefa no projeto.
req_j	Conjunto de habilidades requeridas para a realização da tarefa j.
$\mathscr{T}_j^{c\_est\_t}$	Custo estimado total para realização da tarefa, no início do projeto.
$\mathscr{T}_j^{c\_gasto}(t_l)$	Custo gasto na tarefa $\mathscr{T}_j$ do início do projeto até t_l.
$\mathscr{T}_j^{c\_est\_r}(t_l)$	Custo estimado restante para o término de $\mathscr{T}_j$, a partir de t_l.
$\mathscr{T}_j^{acabou}(t_l)$	Booleano que indica se $\mathscr{T}_j$ já está finalizada em t_l.
$\mathscr{T}_j^{disponivel}(t_l)$	Booleano que indica se $\mathscr{T}_j$ está disponível em t_l.
$\mathscr{T}_j^{tam\_time}(t_l)$	Quantidade de funcionários associados a $\mathscr{T}_j$ em t_l.
$\mathscr{T}_j^{maxfunc}$	Quantidade máxima de funcionários na tarefa $\mathscr{T}_j$.

Fonte: Próprio autor.

2.3.4 Representação da solução

Uma solução do DSPSP é uma matriz de dedicação de tamanho $|M| \times N(t_l)$, formada em um determinado instante $t_l \geq t_0$. No instante t_l, o elemento x_{ij}, da DM, denotado por $x_{ij}(t_l)$, indica a dedicação, em porcentagem, do funcionário e_i na tarefa $\mathscr{T}_j$, no instante t_l. Devido ao **Evento 1** e ao **Evento 2**, relativos à disponibilidade de funcionários, se $e_i^{disponivel}(t_l) = 0$, então $x_{ij}(t_l) = 0 \ \forall j = \{1,2,...,N(t_l)\}$. Da mesma forma, se a tarefa não está disponível, isto é, $\mathscr{T}_j^{disponivel}(t_l) = 0$, então $x_{ij}(t_l) = 0, \ \forall i = \{1,2,...,|M|\}$. Assim, o processo de busca de soluções no instante t_l deve considerar apenas os valores $x_{ij}(t_l) \in \{x_{ij}(t_l) \mid e_i^{disponivel}(t_l) = 1 \wedge \mathscr{T}_j^{disponivel}(t_l) = 1\}$.

2.3.5 Função Objetivo

Shen et al. (2016) estabeleceram quatro objetivos a serem otimizados simultaneamente, listados na Tabela 3 e detalhados a seguir. Em qualquer instante $t_l \geq t_0$, as informações disponíveis são: o conjunto de funcionários disponíveis $e\_disp(t_l)$; as tarefas disponíveis e seus respectivos custos restantes; o GDA $G(V,E)$, atualizado em t_l.

Assumindo que $\mathscr{T}_j^{c\_gasto}(t_l)$ é a quantidade de recurso gastos na tarefa $\mathscr{T}_j$ de t_0 até t_l, o custo estimado restante para esta tarefa, em t_l, é computado por $\mathscr{T}_j^{c\_est\_r}(t_l) = \mathscr{T}_j^{c\_est\_t} - \mathscr{T}_j^{c\_gasto}(t_l)$. Caso $\mathscr{T}_j^{c\_est\_t} \leq \mathscr{T}_j^{c\_gasto}(t_l)$, mas $\mathscr{T}_j$ não está finalizada em t_l, ou seja, $\mathscr{T}_j^{acabou}(t_l) = 0$, então o custo estimado total de $\mathscr{T}_j$ será reavaliado por uma constante B, a partir de uma distribuição normal $\phi(\mu_j, \sigma_j)$ até que a condição $B > \mathscr{T}_j^{c\_gasto}$ seja satisfeita. Só então, atribui-se $\mathscr{T}_j^{c\_est\_t} = B$ e $\mathscr{T}_j^{c\_est\_r}(t_l) = \mathscr{T}_j^{c\_est\_t} - \mathscr{T}_j^{c\_gasto}$. A Tabela 3 sumariza os objetivos a serem otimizados, detalhados no que se segue:

Tabela 3 – Objetivos do DSPSP.

Objetivo	Descrição
$f_1(t_l)$	Duração do projeto no instante t_l
$f_2(t_l)$	Custo inicial do projeto em relação as tarefas disponíveis em t_l
$f_3(t_l)$	Robustez do projeto em relação ao princípio de incerteza de tarefas.
$f_4(t_l)$	Estabilidade do projeto em relação ao escalonamento inicial e em t_l.

Fonte: Próprio autor.

O objetivo $f_1(t_l)$ representa o tempo decorrido necessário para completar todos os recursos restantes alocados em cada tarefa disponível, reescalonada em t_l:

$$f_1(t_l) = duracao_I = \max_{\{j \mid \mathscr{T}_j \in \mathscr{T}\_disp(t_l)\}} (\mathscr{T}_j^{fim}(t_l)) - \min_{\{j \mid \mathscr{T}_j \in \mathscr{T}\_disp(t_l)\}} (\mathscr{T}_j^{ini}(t_l)) \tag{2.1}$$

onde o subscrito I denota o cenário inicial, o qual considera o recurso restante $\mathscr{T}_j^{c\_est\_r}$ como exatamente o recurso necessário para completar $\mathscr{T}_j$ em t_l. Para cada tarefa disponível em t_l,

apenas os recursos restantes são considerados, descartando o recursos já gastos até o determinado instante ($\mathscr{T}_j^{c\_gasto}(t_l)$). Deste modo, $\mathscr{T}_j^{ini}$ representa o instante em que $\mathscr{T}_j$ começou a ser processada a partir de t_l, ao invés de representar o instante do processamento da tarefa em si. Portanto, $\mathscr{T}_j^{ini}(t_l) \geq t_l$, e $\mathscr{T}_j^{fim}(t_l)$ é o instante em que $\mathscr{T}_j$ é finalizada, no escalonamento em t_l.

O objetivo $f_2(t_l)$ representa o custo do escalonamento, isto é, o total de despesas gastas em todos os funcionários disponíveis, por suas dedicações às tarefas disponíveis em t_l, sem considerar a variância dos custos das tarefas (**Evento 4**). Assumindo $t^{'}$ (computado na Equação 2.15, expressa na página 34) como sendo um instante qualquer no projeto depois de t_l, e $\mathscr{T}\_ativa(t^{'})$ denotando o conjunto de tarefas ativas (sendo desenvolvidas) em $t^{'}$, onde uma tarefa é ativa se e somente se ela não possui nenhum predecessor no GDA:

$$f_2(t_l) = custo_I = \sum_{t^{'} \geq t_l} \sum_{e_i \in e\_disp(t_l)} e\_custo_i^{t^{'}}, \tag{2.2}$$

onde $e\_custo_i^{t^{'}}$ significa o custo gasto com o funcionário e_i no instante $t^{'}$. Se a soma das dedicações dos funcionários em todas as tarefas ativas no instante $t^{'}$ for menor ou igual a 1, ou seja, $\sum_{j \in \mathscr{T}\_ativa(t_l)} \leq 1$, têm-se que:

$$e\_custo_i^{t^{'}} = e_i^{salario} \cdot t^{'} \cdot \sum_{j \in \mathscr{T}\_ativa(t^{'})} x_{ij}(t_l), \tag{2.3}$$

Caso contrário, se $1 < \sum_{j \in \mathscr{T}\_ativa(t_l)} x_{ij}(t_l) \leq e_i^{maxded}$, então

$$e\_custo_i^{t^{'}} = e_i^{salario} \cdot t^{'} + e_i^{exc\_salario} \cdot t^{'} \cdot \left(\sum_{j \in \mathscr{T}\_ativa(t^{'})} x_{ij}(t_l) - 1 \right). \tag{2.4}$$

O objetivo $f_3(t_l)$ representa a robustez, a qual avalia a sensibilidade do escalonamento diante da variância dos custos das tarefas:

$$\begin{aligned} f_3(t_l) = & \sqrt{\frac{1}{\eta} \sum_{q=1}^{\eta} \left(max\left(0, \frac{duracao_q(t_l) - duracao_I(t_l)}{duracao_I(t_l)}\right)\right)^2} \\ & + \lambda \sqrt{\frac{1}{\eta} \sum_{q=1}^{\eta} \left(max\left(0, \frac{custo_q(t_l) - custo_I(t_l)}{custo_I(t_l)}\right)\right)^2}, \end{aligned} \tag{2.5}$$

onde $duracao_I$ e $custo_I$ são os valores computados nas equações 2.1 e 2.2, respectivamente. O cálculo da robustez é baseado na variância dos dois objetivos anteriores, e dá-se da seguinte forma: um conjunto de amostras, de tamanho η (definido como parâmetro de entrada), é usado para gerar novos valores de duração e custo para o escalonamento formado em t_l. Cada amostra possui um valor para a duração ($duracao_q$) e custo ($custo_q$) do escalonamento, que são computados da mesma forma que os respectivos objetivos anteriores, a partir de seu custo total $\mathscr{T}_j^{c\_tot_q}$ e dos

recursos gastos $\mathscr{T}_j^{c\_gasto}(t_l)$. O custo total $\mathscr{T}_j^{c\_tot_q}$ de uma amostra, por sua vez, é computado a partir da distribuição normal $\phi(\mu_j, \sigma_j)$ de forma aleatória até que $\mathscr{T}_j^{c\_tot_q} > \mathscr{T}_j^{c\_gasto}(t_l)$. Assim, define-se $\mathscr{T}_j^{c\_rest_q}(t_l) = \mathscr{T}_j^{c\_tot_q} - \mathscr{T}_j^{c\_gasto}(t_l)$ como o custo restante da amostra Θ_q, da tarefa $\mathscr{T}_j$.

Considerando que eficiência é um fator comum em todos os projetos de software, Shen et al. (2016) penalizam o aumento da duração e do custo, pois estes causam a deterioração da eficiência, enquanto que a variância dos mesmos, à medida que diminuem, é truncada pela função máximo. O símbolo λ é um parâmetro de peso, definido na entrada, que captura a importância relativa da sensibilidade do custo do projeto sobre a sensibilidade da duração do mesmo, em relação às incertezas dos custos das tarefas.

O objetivo $f_4(t_l)$ denota a estabilidade do escalonamento, medindo o desvio entre o novo escalonamento e o anterior. Esse objetivo é definido como a soma ponderada dos desvios das dedicações dos funcionários, visando minimizar a troca dos funcionários entre as tarefas. É calculado para todas as tarefas disponíveis em $t_l (t_l > t_0)$, alocadas no escalonamento em t_{l-1}:

$$f_4(t_l) = \sum_{\{i|e_i \in e\_disp(t_{l-1}) \cap e\_disp(t_l)\}} \sum_{\{j|\mathscr{T}_j \in \mathscr{T}\_disp(t_{l-1}) \cap \mathscr{T}\_disp(t_l)\}} w_{ij} |x_{ij}(t_l) - x_{ij}(t_{l-1})|, \tag{2.6}$$

sendo w_{ij} um peso definido da forma:

$$w_{ij} = \begin{cases} 2,0, \ \textit{se } x_{ij}(t_{l-1}) = 0 \ \textit{e } x_{ij}(t_l) > 0, \\ 1,5, \ \textit{se } x_{ij}(t_{l-1}) > 0 \ \textit{e } x_{ij}(t_l) = 0, \\ 1,0, \ \textit{caso contrário}. \end{cases} \tag{2.7}$$

No primeiro caso, a maior penalidade $(2,0)$ é dada toda vez que um funcionário é alocado para uma tarefa, em um novo escalonamento. Isso acontece, pois, o funcionário talvez leve um tempo para se familiarizar com a tarefa, o que pode gerar atrasos. Uma penalidade média $(1,5)$ é dada quando um funcionário passa a não trabalhar mais em uma certa tarefa, o que seria um desperdício de recurso e tempo, uma vez que ele poderia ter recebido um treino para a mesma. Por fim, caso o funcionário continue trabalhando na mesma tarefa com uma dedicação diferente, é dada apenas uma pequena penalização $(1,0)$.

2.3.6 Restrições do problema

Em uma solução do DSPSP, três restrições devem ser respeitadas para que a solução seja considerada válida. A **Restrição 1** e a **Restrição 2**, detalhadas a seguir, são restrições que devem ser obrigatoriamente respeitadas – *hard constraints*, enquanto que a **Restrição 3** é uma restrição que, caso quebrada, a solução em questão receberá uma penalização em seus objetivos – *soft constraint*.

Restrição 1 - Carga de trabalho: No momento $t^{'}$ posterior ao escalonamento ocorrido no instante t_l, a dedicação de um funcionário disponível em todas as tarefas ativas não deve exceder sua dedicação máxima e_i^{maxded} ao projeto. Seja $e\_trab_i^{t^{'}}$ a função de trabalho do funcionário e_i no instante $t^{'}$, computada a partir da soma da suas dedicações a todas as tarefas $\mathscr{T}_j \in \mathscr{T}\_ativa(t^{'})$, então:

$$e\_trab_i^{t^{'}} = \sum_{j \in \mathscr{T}\_ativa(t^{'})} x_{ij}(t_l), onde\ e\_trab_i^{t^{'}} \leq e_i^{maxded} \tag{2.8}$$

Restrição 2 Habilidades dos funcionários: Os funcionários associados à tarefa $\mathscr{T}_j$ no instante t_l devem, coletivamente, possuir todas as habilidades necessárias para a tarefa. Em outras palavras, o conjunto formado a partir da união das habilidades dos funcionários associados a $\mathscr{T}_j$ deve conter ou ser o próprio conjunto de habilidades requeridas para a execução de $\mathscr{T}_j$:

$$\forall\ \mathscr{T}_j \in \mathscr{T}_{disp}(t_l)\ tal\ que\ req_j \subseteq \bigcup_{e_i \in e\_disp(t_l)} \{hab_i \mid x_{ij}(t_l) > 0\} \tag{2.9}$$

Restrição 3 Máximo de funcionários em uma tarefa: O tamanho do time $\mathscr{T}_j^{tamTime}$, isto é, de funcionários associados a tarefa $\mathscr{T}_j$ não deve superar o limite $\mathscr{T}_j^{maxFunc}$, calculado a partir da fórmula (CHANG et al., 2008): $\mathscr{T}_j^{maxFunc} = max\left(1, round\left(2/3 \times (\mathscr{T}_j^{c\_est\_t})^{0,672}\right)\right)$, onde *max* compara dois valores e escolhe o maior, e *round* arredonda o valor para cima. Caso o limite seja excedido e não seja possível reduzir o número de funcionários sem que a **Restrição 2** seja violada, uma penalidade é atribuída ao preço de $\mathscr{T}_j$. Supondo que a quantidade mínima de funcionários necessários que devem juntar-se à tarefa para satisfazer a **Restrição 2** seja $\mathscr{T}_j^{minNum_{fun}}$, então têm-se que o tamanho do time da tarefa $\mathscr{T}_j$ deve ser o máximo entre a quantidade mínima de funcionários e o limite calculado:

$$\forall\ \mathscr{T}_j \in \mathscr{T}\_disp(t_l), \mathscr{T}_j^{tamTime}(t_l) \leq max(\mathscr{T}_j^{maxFunc}, \mathscr{T}_j^{minNum_{fun}}(t_l)) \tag{2.10}$$

2.3.7 Tratamento das Restrições

2.3.7.1 Carga de Trabalho

Se o funcionário e_i excede a sua dedicação máxima, isto é, $e\_trab_i^{t^{'}} > e_i^{maxded}$, então sua dedicação a essas tarefas é dividida por $e\_trab_i^{t^{'}}$. O valor normalizado da dedicação $x_{ij}(t_l)$ passa a ser denotado por $d_{ij}(t_l) = x_{ij}(t_l)/max(1, e\_trab_i^{t^{'}}/e_i^{maxded})$. Dessa forma, normalizando as dedicações que ultrapassam o limite de dedicação do funcionário e_i, ninguém irá trabalhar acima do seu limite, e a carga de trabalho será respeitada.

2.3.7.2 Habilidades dos funcionários

O tratamento dessa restrição leva em consideração a proficiência de cada funcionário nas tarefas ao validar um escalonamento e lidar com esta restrição. Dessa forma, calcula-se a dedicação total ajustada $A\_\mathscr{T}_j(t_l)$, para a tarefa $\mathscr{T}_j$, da forma descrita a seguir:

Inicialmente, é calculada a dedicação total $Td_j(t_l)$ de todos os funcionários disponíveis em $\mathscr{T}_j$:

$$Td_j(t_l) = \sum_{e_i \in e\_disp(t_l)} d_{ij}(t_l). \tag{2.11}$$

Em seguida, a aptidão total $F_j(t_l)$ de todos os funcionários disponíveis em $\mathscr{T}_j$ é computada:

$$F_j(t_l) = \left(\sum_{e_i \in e\_disp(t_l)} e_{ij}^{proficiencia} \cdot d_{ij}(t_l) \right) / Td_j(t_l), \tag{2.12}$$

onde $F_j(t_l)$ é uma fração da dedicação total gasta pelos funcionários na tarefa $\mathscr{T}_j$. Esse cálculo é necessário, pois, apesar dos funcionários possuírem uma dedicação de $d_{ij}(t_l)$ na tarefa $\mathscr{T}_j$, se as suas proficiências forem baixas, $\mathscr{T}_j$ irá levar mais tempo para terminar, da mesma forma se as dedicações dos funcionários fossem menor que $d_{ij}(t_l)$. Assim, a Equação 2.12 reduz as dedicações baseada nas proficiências.

O valor, $F_j(t_l)$ é, então, convertido para o valor de custo unitário $V_j(t_l)$ (CHANG et al., 2008):

$$C_j(t_l) = max(1, 8 - round(F_j(t_l) \cdot 7 + 0,5)), \tag{2.13}$$

onde o valor de $C_j(t_l)$ varia no intervalo $[1,7]$. Caso $C_j(t_l) = 1$, então os funcionários escalonados para $\mathscr{T}_j$ são os mais adequados para ela. Finalmente, a dedicação total ajustada é obtida como sendo:

$$A\_\mathscr{T}d_j(t_l) = Td_j(t_l)/V_j(t_l). \tag{2.14}$$

Com esses valores computados, e assumindo $\mathscr{T}_j^{c\_rest}$ como o custo restante para o término de $\mathscr{T}_j$, é calculado o valor de $t^{'}$ da seguinte forma:

$$t^{'} = \min_{\mathscr{T}\_ativa(t^{'})} (\mathscr{T}_j^{c\_rest}/A\_\mathscr{T}d_j(t_l)) \tag{2.15}$$

Se no instante t_l algum candidato não possuir alguma habilidade necessária para alguma tarefa a qual ele foi alocado, então uma penalidade muito alta é atribuída para cada objetivo

dessa solução. Assumindo *reqsk* como a quantidade de habilidades faltantes, os objetivos são penalizados da seguinte forma:

$$\begin{aligned} f_1(t_l) &= reqsk \cdot 2 \cdot \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} \mathscr{T}_j^{c\_est\_r}(t_l) \Bigg/ \left(\min_{e_i \in e\_disp(t_l)} e_i^{maxded} / k / \max_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} V_j \right) \\ &= reqsk \cdot 2 \cdot \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} \mathscr{T}_j^{c\_est\_r}(t_l) \Bigg/ \left(\min_{e_i \in e\_disp(t_l)} e_i^{maxded} / k / 7 \right) \\ &= reqsk \cdot 14k \cdot \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} \mathscr{T}_j^{c\_est\_r}(t_l) \Bigg/ \min_{e_i \in e\_disp(t_l)} e_i^{maxded}, \end{aligned} \tag{2.16}$$

$$\begin{aligned} f_2(t_l) &= reqsk \cdot 2 \cdot \sum_{e_i \in e\_disp(t_l)} \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} e_i^{exc\_salario} \cdot \mathscr{T}_j^{c\_est\_r}(t_l) \cdot 7 \\ &= reqsk \cdot 14 \cdot \sum_{e_i \in e\_disp(t_l)} \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} e_i^{exc\_salario} \cdot \mathscr{T}_j^{c\_est\_r}(t_l), \end{aligned} \tag{2.17}$$

$$f_3(t_l) = reqsk \cdot 2 \cdot C_{rob}, \tag{2.18}$$

$$f_4(t_l) = reqsk \cdot 2 \cdot |e\_disp(t_l)| \cdot |\mathscr{T}\_disp(t_l)| \cdot \max_{e_i \in e\_disp(t_l)} e_i^{maxded}, \tag{2.19}$$

onde C_{rob} e k são parâmetros de entrada. Os valores dos objetivos após a penalização são maiores do que qualquer escalonamento possível, uma vez que:

- A duração do projeto é $7k \cdot \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} \mathscr{T}_j^{c\_est\_rst}(t_l) / \min_{e_i \in e\_disp\_set(t_l)} e_i^{maxded}$, no máximo. Isso acontece, pois, no pior caso as tarefas são executadas uma a uma. A dedicação para cada tarefa é sempre $\min_{e_i \in e\_disp(t_l)} e_i^{maxded}/k$ e o valor de custo unitário para cada tarefa é o máximo (7).
- O custo é penalizado de acordo com o salário dos funcionários quando estes excedem sua carga de trabalho. Assim, a dedicação total de cada funcionário para cada tarefa é igual ao custo estimado restante para finalizar a tarefa, $\mathscr{T}_j^{c\_est\_r} \cdot 7$, onde 7 é o máximo obtido pelo valor de custo unitário. Esse valor é similar ao cenário onde todos os funcionários são os únicos a trabalhar na tarefa. Portanto, o valor total do custo é sempre $\sum_{e_i \in e\_disp(t_l)} \sum_{\mathscr{T}_j \in \mathscr{T}\_disp(t_l)} e_i^{exc\_sal} \cdot \mathscr{T}_j^{c\_est\_r}(t_l) \cdot 7$, no máximo.
- Como parâmetro de entrada, o valor da constante C_{rob} deve ser grande o suficiente para superar o valor da robustez.
- No pior caso, o desvio das dedicações para cada tarefa disponível é $\max_{e_i \in e\_disp(t_l)} e_i^{maxded}$. Logo, o valor da estabilidade de um projeto é sempre, no máximo, $|e\_disp(t_l)| \cdot |\mathscr{T}\_disp(t_l)| \cdot \max_{e_i \in e\_disp(t_l)} e_i^{maxded}$.

Por fim, todos os objetivos crescem proporcionalmente à quantidade de habilidades faltantes, *reqsk*. Isso significa que as penalidades diminuem sempre que o número de habilidades faltantes diminuem.

2.3.7.3 Máximo de Funcionários em uma Tarefa

Duas heurísticas são utilizadas para diminuir a quantidade de funcionários em uma tarefa $\mathscr{T}_j$. A primeira consiste em setar para $x_{ij}(t_l) = 0$ a dedicação do funcionário e_i para a tarefa $\mathscr{T}_j$ caso ele não possua todas as habilidades necessárias para a tarefa, ou seja, $e_{ij}^{proficiencia} = 0$. A segunda é verificar se para cada tarefa $\mathscr{T}_j \in \mathscr{T}\_disp(t_l)$, $\mathscr{T}_j^{tamTime} > \mathscr{T}_j^{maxFunc}$. Caso seja verdade, o seguinte procedimento é executado: o conjunto dos funcionários, em cada tarefa $\mathscr{T}_j$, é ordenado por $e_{ij}^{proficiencia}$. A partir do funcionário e_i com menor proficiência, é verificado se a remoção dele viola a restrição de habilidades necessárias para a tarefa $\mathscr{T}_j$. Caso não viole, então o funcionário e_i é removido da tarefa $\mathscr{T}_j$, ou seja $x_{ij}(t_l) = 0$. Essa verificação é feita para os funcionário subsequentes no conjunto, em todas as tarefas. Caso a quantidade de funcionários não consiga ser reduzida até $\mathscr{T}_j^{maxFunc}$, então uma penalidade é aplicada no preço da tarefa $\mathscr{T}_j$, segundo a equação 2.20.

$$\mathscr{T}_j^{cst} = \mathscr{T}_j^{cst} \cdot \left(1 + \frac{\mathscr{T}_j^{tamTime} \cdot (\mathscr{T}_j^{tamTime} - 1)/2}{Z}\right), \tag{2.20}$$

onde Z é um parâmetro de entrada e $\mathscr{T}_j^{cst}$ é o custo de $\mathscr{T}_j$ antes da penalidade.

2.4 Inserção do Modelo Dinâmico no Ciclo de Desenvolvimento de Software

Após a ocorrência de um evento dinâmico, o processo de busca é ativado e novas matrizes de dedicações, que respeitam todas as restrições do modelo, são geradas e apresentadas ao líder do projeto. Este, por sua vez, analisa as matrizes e seleciona aquela que maximiza os objetivos de acordo com as necessidades (o projeto tem uma menor duração priorizada, por exemplo). A matriz de dedicação escolhida seria adotada para o projeto e todos os funcionários passam a trabalhar de acordo. Esse processo ocorre toda vez que ocorre um evento dinâmico e o projeto não esteja finalizado. Após a adoção de um escalonamento, torna-se possível a criação de novos *charts* do projeto, como o diagrama de Gantt ou de Fluxo.

Entretanto, pode não ser prático que esse processo ocorra de forma manual. O conjunto de soluções retornado pode ser grande e uma análise a partir de um humano pode ser impraticável devido devido, por exemplo, à ocorrência de novas restrições de prazo para a conclusão do projeto. Dessa forma, um seletor automatizado de escalonamentos é detalhado na Seção 3.2, o qual se baseia na importância definida entre os objetivos para selecionar um escalonamento

dentre vários. O seletor dá liberdade ao líder do projeto para determinar quais objetivos serão priorizados, de acordo com as necessidades do momento. Por exemplo, podem haver momentos em que minimizar a duração do projeto seja a prioridade, e após a ocorrência de novas alterações ou acordos, minimizar o custo passe a ser o principal objetivo.

2.5 O DSPSP no contexto da Engenharia de Software Baseada em Busca

A Engenharia de Software Baseada em Busca – *Search-Based Software Engineering* foi introduzida em Harman e Jones (2001). A SBSE investiga técnicas de buscas, como meta-heurísticas, para a solução de problemas da engenharia de software, muitos deles resolvidos de forma *ad hoc*. Desde sua concepção, a SBSE vem crescendo e atraindo mais pesquisas sobre novas modelagens e novas técnicas de busca de soluções (HARMAN, 2012). De acordo com (HARMAN; JONES, 2001), apenas dois quesitos são necessários para estudar um problema da engenharia de software na área de SBSE:

1. Uma representação formal do problema.
2. A definição do quão boa é uma solução através dos valores do(s) objetivo(s) (*fitness*) .

O item 1 refere-se a como o problema pode ser representado de tal forma que possa ser aplicado técnicas computacionais, tais como algum cálculo sobre a representação. São exemplos de representações um vetor, ou uma matriz. Já a função objetivo, item 2, é responsável por guiar o processo de busca, uma vez que é ela quem quantifica, através de métricas, o quão boa é a solução em relação a um referencial, que pode ser simplesmente o conjunto disponível de soluções.

A modelagem proposta por Shen et al. (2016), objeto da seção 2.2, para o Problema do Escalonamento Dinâmico em Projeto de Software se encaixa nos critérios 1 e 2 da SBSE. A representação formal permite gerar a matriz de dedicação, como uma solução, e a partir da mesma, os *fitness* são representados pelos quatro objetivos: duração, custo, estabilidade e robustez. A busca por soluções se dá pela aplicação de meta-heurísticas, descritas a seguir.

2.6 Meta-heurísticas

O Problema do Escalonamento Dinâmico é um problema de otimização, onde, dentre todos os objetivos, deseja-se otimizá-los de tal forma que, escolhidas suas variáveis de decisão, sejam obtida soluções que apresentem um bom compromisso entre eles. No entanto, problemas de otimização impõem um grande desafio para a computação, sobretudo no que se refere a complexidade para oferecer soluções para tais problemas.

Desde que muitos problemas de otimização, como o SPSP (CRAWFORD et al., 2014), são classificados na literatura como pertencentes à classe $\mathscr{NP}$-Difícil, onde até o momento não há evidências de algoritmos exatos que possam resolvê-los sem consumir um tempo (ou memória) que seja exponencial em relação ao tamanho da entrada do mesmo. Diante do esforço em resolver problemas dessa complexidade com um tempo de execução viável, as heurísticas são aplicadas. Assim sendo, define-se heurísticas da seguinte forma:

Definição 2.6.1 (Heurísticas) *Uma heurística é uma técnica de solução de problema onde a solução parcial mais apropriada no estado atual é selecionada a partir de métodos que empregam regras comparativas. (COELLO; LAMONT; VELDHUIZEN, 2007)*

Segundo (GOLDBARG; GOLDBARG; LUNA, 2016), as heurísticas visam alcançar soluções avaliadas como aceitáveis para um dado problema utilizando um esforço computacional consideravelmente aceitável, garantindo, em determinadas condições, a otimalidade da solução gerada. Isso acontece, pois, os métodos heurísticos sistematicamente selecionam soluções aparentemente boas, segundo seus métodos de seleções empregados. Soluções boas para tais métodos são aquelas que, localmente, se destacaram em relação às outras, e são interessantes de serem exploradas. Esses métodos são oriundos de várias inspirações, podendo ser uma simples comparação entre as soluções candidatas ou até mesmo a computação de alguma métrica estatística qualquer. Mas, devido a enorme quantidade de problemas e variantes destes na natureza, um conjunto de heurísticas pode ser bom para uma classe de problema mas péssimo para outros (LAM; LI, 2010).

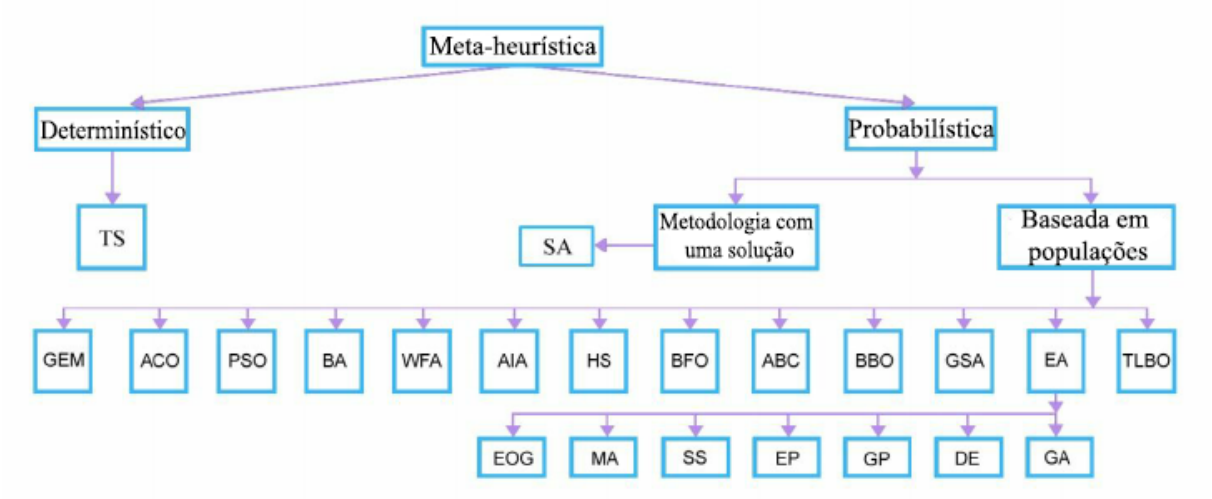
A busca por algoritmos que possuem um maior poder de generalização resultaram em métodos abstratos que estão em um nível maior do que o problema ao qual será aplicado. Esses algoritmos receberam o nome de *meta-heurísticas* e representam o atual estado da arte de otimização (GOLDBARG; GOLDBARG; LUNA, 2016). O prefixo *meta* sugere uma abstração do algoritmo na solução do problema. De fato, segundo (GOLDBARG; GOLDBARG; LUNA, 2016), uma meta-heurística pode ser vista como método que guia o processo de computação de uma ou mais heurísticas para a solução de um problema (apud DORIGO; STÜZLE, 2004). Meta-heurísticas são, então, definidas da seguinte forma:

Definição 2.6.2 (Meta-heurísticas) *Meta-heurísticas são estratégias de alto nível que guiam outros processos de baixo nível, como heurísticas, buscando soluções viáveis em contextos com domínios complexos. (COELLO; LAMONT; VELDHUIZEN, 2007)*

Apesar de serem algoritmos abstratos e de fácil adaptação no geral, nenhuma meta-heurística é boa para todos os tipos de problema. A Figura 10 elenca algumas categorias e subcategorias, as quais possuem diversas meta-heurísticas relacionadas. Mas, de acordo com o teorema *No Free Lunch* (WOLPER; MACREADY, 1997), todas as metas-heurísticas possuem

desempenho médio igual se comparadas com todos os problemas. Isso implica que meta-heurísticas são estaticamente iguais em soluções de problemas computacionais (LAM; LI, 2010).

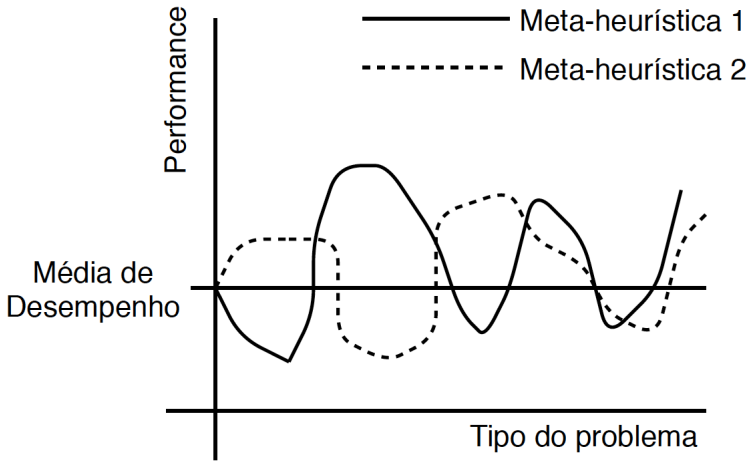
Figura 10 – Diversas categorias e subcategorias de meta-heurísticas.



Fonte: Adaptado de (KURADA; PAVAN; RAO, 2013).

Em outras palavras, Ho e Pepyne (2002) explicam que uma meta-heurística de propósito geral é impossível de ser modelada e que a única maneira de uma meta-heurística se sobressair em relação a uma outra é comparando-as em apenas problemas específicos. A Figura 11 mostra um possível comportamento de duas meta-heurísticas quando aplicadas a vários problemas. Os resultados apresentados por esses teoremas implicam que é importante a investigação de várias meta-heurísticas para um dado problema, sobretudo se esse for de notável importância.

Figura 11 – Desempenho de duas meta-heurísticas quando aplicadas a diversos problemas.



Fonte: Próprio autor.

2.6.1 Meta-heurísticas Evolucionárias

As meta-heurísticas usadas neste trabalho são consideradas pertencentes à classe de meta-heurísticas evolucionárias – *Multiobjective Evolutionary Algorithms* (MOEA). Essa categoria de algoritmos possui inspirações oriundas de fenômenos da natureza, tais quais biológico, químico, etc (GOLDBARG; GOLDBARG; LUNA, 2016). É característico dessa classe basear o conjunto solução como uma população, onde cada indivíduo dessa população é uma solução por si própria. O método de busca das meta-heurísticas evolucionárias pode ser facilmente guiado pelo domínio do problema sem que precise ser modificado, isto é, sem que o mesmo tenha de ser alterado para que seja exexecutado na meta-heurística (COELLO; LAMONT; VELDHUIZEN, 2007). Esse aspecto transformou-se em uma vantagem para os MOEA, visto que por conta disso seu desenvolvimento se tornou simples, bem como seu entendimento e sua visualização. Harman (2012) cita que MOEA é o tipo de meta-heurística mais usado na SBSE. Portanto, todas os MOEA compartilham um esquema algorítmico, mostrado no Algoritmo 1, com poucas variações.

Algoritmo 1 Pseudocódigo de uma Meta-heurística Evolucionária

```
1:  função GERA SOLUÇÕES
2:      Seja S o conjunto de soluções, inicialmente vazio
3:      S ← GeraPrimeiraPopulacao(tamanho)
4:      AvaliaPopulacao(S)
5:      CriterioDeParadaAtingido ← falso
6:      enquanto não CriterioDeParadaAtingido faça
7:          S' ← AtualizaPopulacao(S)
8:          AvaliaPopulacao(S')
9:          S ← S'
10:         atualizaCriterioDeParada()
11:     fim enquanto
12:     Seja S a saída desse algoritmo
13: fim função
```

Como pode ser visto no Algoritmo 1, uma MOEA começa o processo de busca criando uma população inicial, P, servindo como um ponto de partida (linha 3). Assim que criada, P passa por um processo de avaliação de seus indivíduos (linha 4) baseado nos objetivos do problema e na dominância de Pareto. Dessa forma, o processo principal da meta-heurística é realizado até que o critério de parada, que é definido geralmente como número de iterações máximo, seja atingido (linha 10). Dentro do laço principal ocorre o uso dos operadores da meta-heurística, bem como a evolução da população. Os operadores são responsáveis pelo procedimento de atualização da população (linha 7), onde usualmente serão aplicados métodos de mutação, seleção e recombinação. Sendo um algoritmo estocástico, é possível que a nova população gerada (linha 7), S, tenha indivíduos piores do que a população atual P. A questão levantada na literatura é sobre como melhor aproveitar a nova população S (COELLO; LAMONT; VELDHUIZEN, 2007). Um método simples é usado no Algoritmo 1 que é atribuir toda a população nova a atual. Assim, até que o critério de parada não seja atingido (linha 6), a meta-heurística irá procurar por

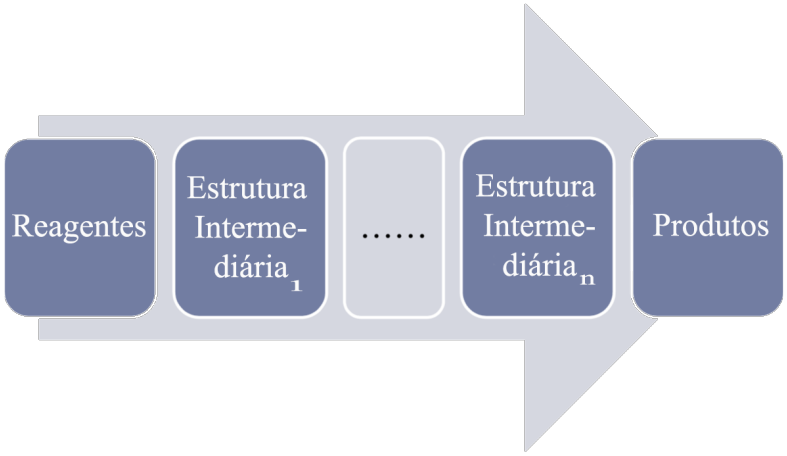
novas populações e ao final irá retorná-la como saída para o usuário.

2.7 Otimizações por Reações Químicas

A *Nondominated Sorting Chemical Reaction Optimization* (NCRO) é uma meta-heurística recentemente proposta em (BECHIKH; CHAABANI; SAID, 2015). Inspirada nas reações químicas, onde as moléculas sempre procuram uma estrutura molecular mais estável, e na lei de Lavoisier, que diz que na natureza nada se cria, nada se perde, tudo se transforma, a NCRO é implementada de forma similar a uma consagrada meta-heurística, a NSGA-II (DEB et al., 2002), largamente utilizada na SBSE. Os mecanismos de busca e estruturas da NCRO foram estendidos a partir de uma meta-heurística criada para problemas com apenas um objetivo, a *Chemical Reaction Optimization* (CRO) (LAM; LI, 2010).

A principal observação é de que reações químicas possui um fluxo direcional (apresentado na figura 12), no qual existe um conjunto de estruturas moleculares no início – produtos –, e que passam por um processo de transformação (variando para diversas estruturas intermediárias), até que seja alcançado um estado estável, contendo apenas estruturas denominadas de reagentes. Todas as estruturas moleculares são mapeadas para a população das meta-heurísticas evolcionárias. Assim, o conjunto dos reagentes é a primeira população, enquanto que os produtos são as soluções que compõem a fronteira de pareto retornada.

Figura 12 – Direção de uma reação química, com início e fim.



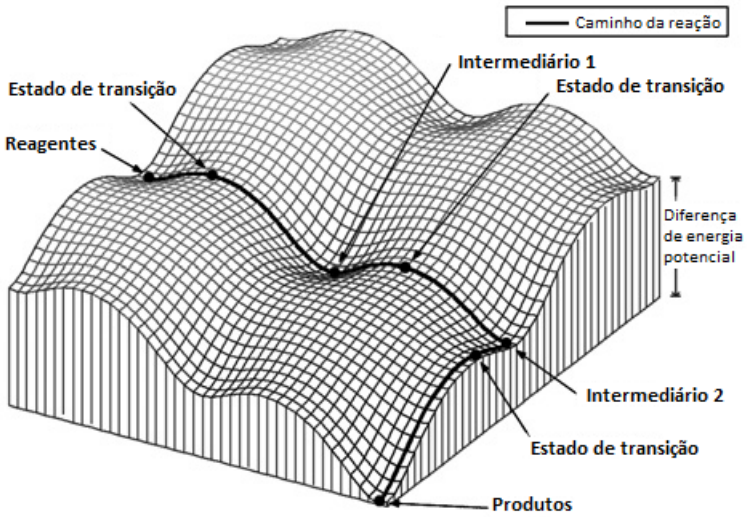
Fonte: Adaptado de (LAM; LI, 2010b?).

2.7.1 Chemical Reaction Optimization

Lam e Li (2010) propouseram a *Chemical Reaction Optimization* - Otimização por reações químicas, a qual também é uma meta-heurística evolucionária, onde cada indivíduo da sua população é análogo a uma molécula, na química. Motivados pelas interações inter e extra-moleculares que ocorrem dentro de uma reação química, a Superfície de Energia Potencial

– *Potential Energy Surface* (PES) permite ao observador verificar o estado de uma molécula, ou um conjunto de moléculas, em um dado momento de uma reação química, bastando apenas fornecer algum ponto na superfície, a qual pode possuir várias dimensões. A Figura 13 ilustra um exemplo de um PES.

Figura 13 – Superfície de Energia Potencial.



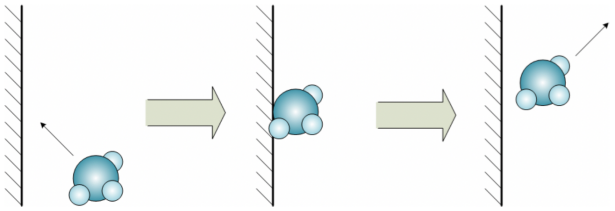
Fonte: Adaptado de (LAM; LI, 2010, p. 382).

A partir do PES, um referencial pode calcular informações sobre os elementos químicos envolvidos na reação, onde cada um é composto por moléculas agitadas, as quais possuem energia potencial (PE) e cinética (KE). O caminho descrito pela linha em negrito no PES da Figura 13 indica que uma reação sempre procura locais com menos energia possível, de tal forma que, assim, os produtos sempre possuem menos energia que os reagentes, portanto são mais estáveis. A energia cinética pode ocasionar uma diferença de potencial maior, facilitando a criação de uma estrutura molecular mais instável que a em questão. Reações químicas possuem um espaço físico limitado, e agitadas, as moléculas podem colidir entre si ou entre os extremos do espaço onde ela estão. Assim, no CRO, uma molécula é representada como uma solução e os eventos (reações) ocasionados pela agitação das moléculas funcionam como os operadores do CRO, para alteração estrutural de uma solução. São quatro operadores, detalhados a seguir:

Reação 1 - Colisão na parede sem efeito: corresponde quando a molécula choca contra a parede do recipiente e nada acontece. A estrutura da molécula, então, é substituída por uma similar. Matematicamente, é quando uma solução se transforma em uma solução que está na vizinhança atual.

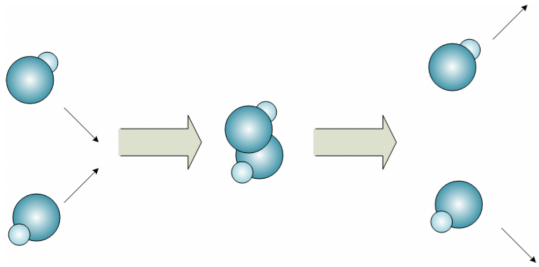
Reação 2 - Colisão intermolecular sem efeito: essa reação acontece quando duas moléculas se chocam. O comportamento é similar ao da **Reação 1**, com exceção de que nessa são duas moléculas, visando explorar a vizinhança simultaneamente, cada uma correspondendo as moléculas substituídas.

Figura 14 – Esquematização da reação de colisão na parede sem efeito.



Fonte: Adaptado dos slides de (LAM; LI, 2010b?)

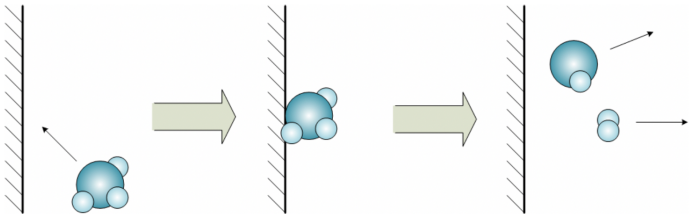
Figura 15 – Esquematização da reação de colisão intermolecular sem efeito.



Fonte: Adaptado dos slides de (LAM; LI, 2010b?)

Reação 3 - Decomposição: acontece quando a molécula choca contra a parede do recipiente e se quebra em várias partes (duas, como mencionado anteriormente). O objetivo da decomposição é fazer com que uma o algoritmo explore locais mais distantes da vizinhança da solução atual.

Figura 16 – Esquematização da reação de decomposição.

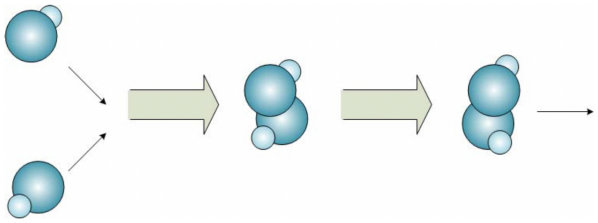


Fonte: Adaptado dos slides de (LAM; LI, 2010b?)

Reação 4 - Síntese: o inverso da **Reação 2**, a síntese acontece quando duas moléculas se chocam e se fundem, resultando em apenas uma nova molécula. A molécula resultante deve ser bem diferente das duas anteriores.

Os dois tipos de energia que uma solução pode possuir, Potencial e Cinética, também fazem parte do processo evolucionário da CRO. Sendo ω uma estrutura molecular no PES e y a função objetivo, a PE é representada pela Equação 2.21, a qual indica o quão boa é aquela

Figura 17 – Esquematização da reação de síntese.



Fonte: Adaptado dos slides de (LAM; LI, 2010b?)

solução. Em problemas de minimização, menores valores são desejáveis. Já a KE, quantifica a capacidade de uma solução escapar de um local mínimo (ou máximo).

$$PE_{\omega} = y(\omega) \tag{2.21}$$

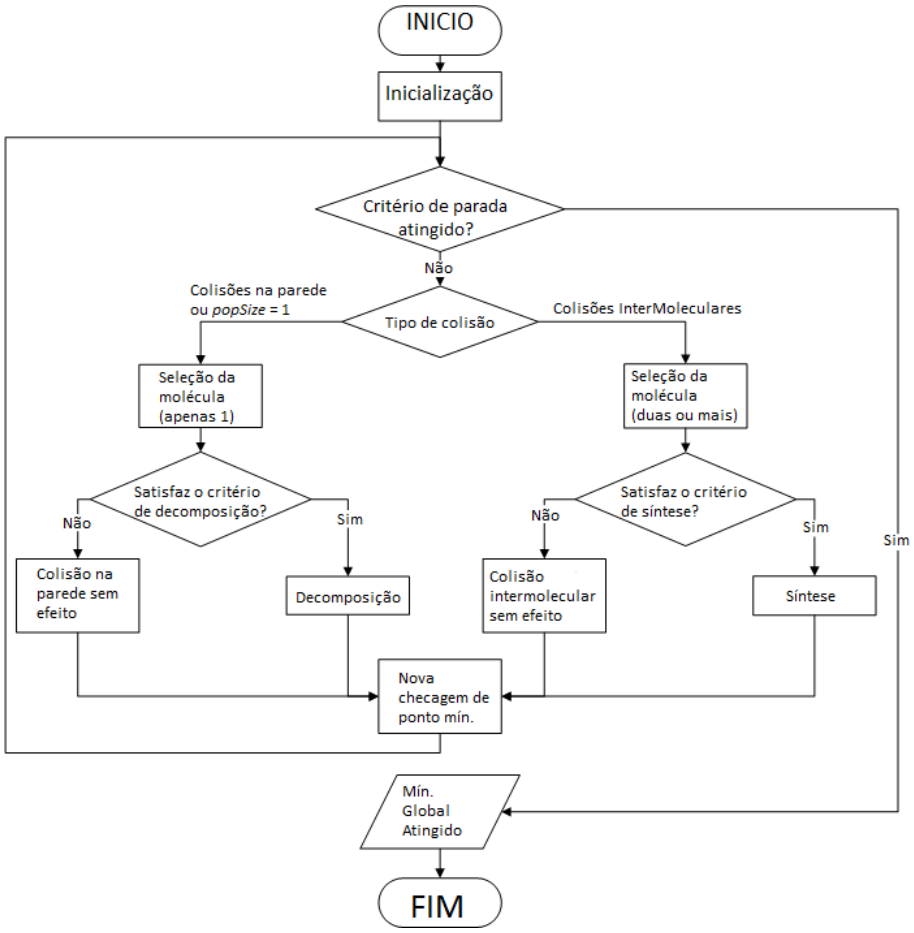
Estruturas moleculares sempre procuram um caminho no PES que resulta em uma estrutura com menor valor de PE. Assim, uma transformação na CRO só é permitido se $PE_{\omega} \geq PE_{\omega'}$. Se a inequação anterior não for satisfeita, então a reação é permitida somente se $PE_{\omega} + KE_{\omega} \geq PE_{\omega'}$. Por simplicidade, uma reação ou envolve apenas uma molécula, ou duas moléculas. Nesse caso, as inequações das duas moléculas somam-se para verificar a validação da transformação. Caso não seja satisfeita, então a(s) estrutura(s) permanecerão inalterada(s). A CRO possui três fases definidas, ilustradas na Figura 18, em sua execução: inicialização, evolução e resultados.

Para a fase da inicialização, são definidos os parâmetros de entrada, descritos pela Tabela 4. Em seguida, a CRO gera sua população inicial ao mesmo tempo em que calcula a PE de cada molécula e associa o valor inicial da KE. Assim, a meta-heurística entra em um laço que irá repetir até que o critério de parada seja atingido. Em cada iteração, um número aleatório t entre o intervalo $[0,1]$ é gerado. Se t for maior do que o parâmetro de entrada *Seletor molecular*, será considerada uma operação com duas moléculas. Caso contrário, $t \leq$ *Seletor molecular* significa uma reação com apenas uma molécula. Após definida a quantidade de moléculas na reação, é determinado qual será o tipo da reação, podendo ser qualquer uma das citadas anteriormente, de acordo com o t. O tipo de reação é definido a partir da estrutura molecular, que verifica se o critério de cada reação é satisfeito, como mostrado na Figura 18. Ao final de uma reação, a CRO verifica se o mínimo (ou máximo) global foi atingido, e o laço chega ao fim.

2.7.2 Nondominated Sorting Chemical Reaction Optimization

A partir dos bons resultados da CRO com problemas com apenas um objetivo (LAM; LI, 2010), Bechikh, Chaabani e Said (2015) apresentaram uma extensão da CRO para problemas multiobjetivos, a *Nondominated Sorting Chemical Reaction Optimization*. Para isso, foram adicionados elementos que pudessem qualificar uma solução a partir de seus respectivos

Figura 18 – Fluxograma do CRO.



Fonte: Adaptado de (LAM; LI, 2010, p. 388).

Tabela 4 – Parâmetros de uso do CRO.

Parâmetro	Descrição
Energia inicial	Energia dentro do *buffer* ao início da execução.
Energia Cinética inicial	Energia Cinética inicial de cada molécula.
Taxa de perda de energia	Porcentagem de perda de energia cinética das moléculas.
Seletor molecular	Limite entre reação uni ou inter-molecular.
Step	Parâmetro controlador das duas colisões sem efeito.
Limiar da Síntese	Limite entre síntese ou colisão sem efeito.
Limiar da Decomposição	Limite entre decomposição ou colisão na parede sem efeito.

Fonte: Próprio autor.

ranques de Pareto, ao mesmo tempo em que a conservação de energia da população não ficasse desbalanceada. Duas principais alterações foram propostas: 1) O valor da energia potencial (PE) e 2) a geração da população descendente, detalhadas a seguir.

Em relação à Energia Potencial, originalmente, o valor associado à esta energia fazia referência apenas ao valor da função objetivo do problema. Mas essa proposta é inválida nos

cenários com muitos objetivos, visto que o valor da energia potencial deve corresponder à qualidade de todos os vetores de decisões do problema. Então, a fórmula da PE é dada por:

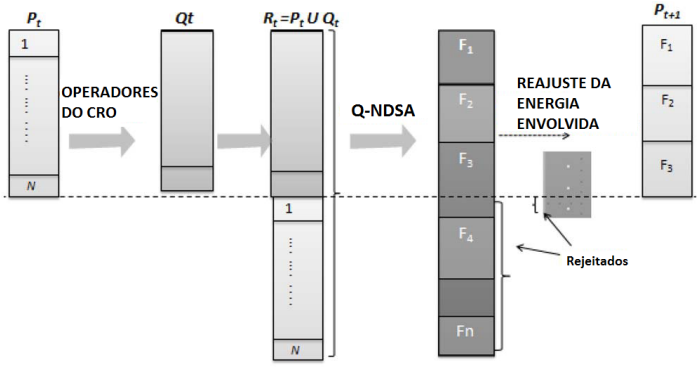
$$PE(x) = rank(x) + e^{-crowd(x))}, \tag{2.22}$$

onde x é a solução, *rank* é o ranque de Pareto ao qual a solução pertence e *crowd* corresponde à distância de *crowding* usada no NSGA-II (DEB et al., 2002).

Já quanto à Geração dos Descendentes, a NCRO se diferencia de outras meta-heurísticas, a exemplo dos algoritmos genéticos, no sentido de que a nova população disputa pela validação com a anterior dentro do procedimento da reação que está ocorrendo, e não em separado. Então, a segunda adaptação consiste em fazer com que essa disputa seja feita da forma descrita a seguir: uma nova população, Q_t, é gerada a partir da aplicação das operações (reações) do CRO na população atual, P_t. O segundo passo consiste em criar outra população, R_t, onde $R_t = P_t \cup Q_t$. Esse procedimento garante o elitismo – método que garante que as melhores soluções não serão descartadas – na população.

Depois desse passo, a população é ordenada segundo o algoritmo Q-NDSA (BECHIKH; CHAABANI; SAID, 2015). Assim, as soluções com menor *fitness* serão as primeiras, e consequentemente, são as melhores soluções. Para restaurar o equilíbrio da energia envolvida na execução, apenas algumas (as melhores) soluções prosseguirão para a próxima população de fato, P_{t+1}. O procedimento é dado a partir da obtenção do valor *fitness*, baseado no cálculo da energia potencial. A Figura 19 ilustra o processo iteracional da NCRO.

Figura 19 – Esquema iteracional básico do NCRO.



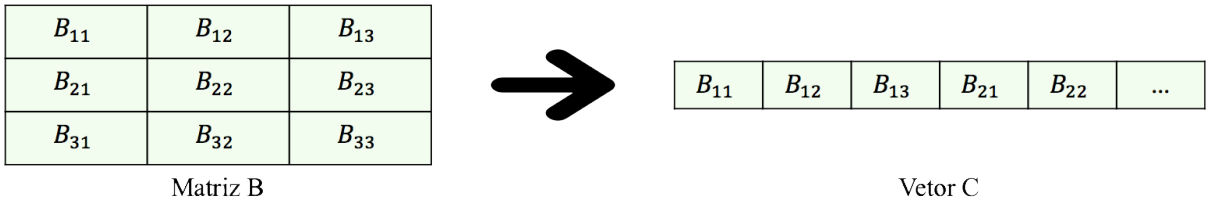
Fonte: Adaptado de (BECHIKH; CHAABANI; SAID, 2015, p. 2055).

2.8 Adaptação do NCRO ao DSPSP

Para que o NCRO seja aplicável ao DSPSP, é ncessário estabelecer os elementos que irão compor a solução do problema. Como já citado na Seção 2.3.4, uma solução para o DSPSP é uma matriz de dedicação. Uma vez que matrizes também são vetores, as soluções do DSPSP serão

convertidas para que seja possível aplicar os operadores evolutivos mais utilizados na literatura. Seja B uma matriz de tamanho $Y \times Z$, então é possível converter B para um vetor C de $Y \times Z$ posições, onde $B_{ij} = C_{i \times Z + j}$ (valores indexados por 0). A Figura 20 ilustra um mapeamento de uma matriz para um vetor.

Figura 20 – Mapeando posições de uma matriz para um vetor. A posição B_{22} foi mapeada para a posição C_4.



Fonte: Próprio autor.

No NCRO, a molécula que representa a solução é composta por diversos atributos, como a KE e a PE. Dessa forma, cada molécula também possuirá uma matriz de dedicação na forma de um vetor, a qual poderá ser alterada por uma reação química, através dos operadores selecionados, onde cada uma dessas reações alteram a matriz de uma forma particular. Desde que as reações acontecem com uma ou duas moléculas, então assume-se duas soluções α e β, já mapeadas para seus respectivos vetores, ilustradas a seguir. Os vetores são alterados da seguinte forma:

Reação 1 - Colisão na parede sem efeito: uma molécula choca-se contra a extremidade do container, alterando a sua estrutura molecular. A figura 21 exibe os efeitos dessa reação no vetor que representa a matriz de dedicação. Observa-se que, após a reação, os valores de α_4 e α_5 foram alterados para X e A, respectivamente.

Figura 21 – Exemplo da Reação 1.

Fonte: Próprio autor.

Reação 2 - Colisão intermolecular sem efeito: duas moléculas chocam-se entre si, resultando em um efeito similar a reação anterior, alterando ambas as estruturas moleculares. É ilustrado na figura 22 que as alterações na primeira molécula foram de α_2 para A, α_5 para B, enquanto que na segunda molécula as ocorreram em β_1, β_3 e β_5 (para D, D, A, respectivamente).

Reação 3 - Decomposição: uma molécula gera duas outras moléculas com estruturas moleculares descendentes de si própria. A ideia é fazer com o que novos indivíduos possam sair do seu local de busca. É possível observar, na figura 23, que, na primeira molécula os

Figura 22 – Exemplo da Reação 2.

Fonte: Próprio autor.

valores alterados foram em α_2 e α_4, passando para X e Y, respectivamente. Já na segunda molécula, α_3 tornou-se Z, enquanto que α_5 passou para A.

Figura 23 – Exemplo da Reação 3.

Fonte: Próprio autor.

Reação 4 - Síntese: duas moléculas chocam-se entre si e formam uma nova molécula com estrutura molecular baseada nas duas anteriores. Essa reação permite que a meta-heurística diversifique o seu espaço de busca. Essa intenção é visualizada na figura 24, onde a molécula gerada possui valores herdados de seus pais. Dessa forma, α_2 e α_5 são provenientes de α, enquanto que β_1, β_3, β_4, são herdados de β.

Figura 24 – Exemplo da Reação 4.

Fonte: Próprio autor.

Tanto α e β são soluções escolhidas de forma aleatória de acordo com o funcionamento do NCRO. Já os novos valores inseridos nas soluções (A, D, X, ...), bem como as posições que serão alteradas, são definidos de acordo com os operadores evolucionários utilizados para desempenhar essas alterações. Os operadores evolucionários escolhidos para esse trabalho são descritos mais detalhadamente na Seção 3.3.

2.9 Nondominated Sorting Genetic Algorithm II

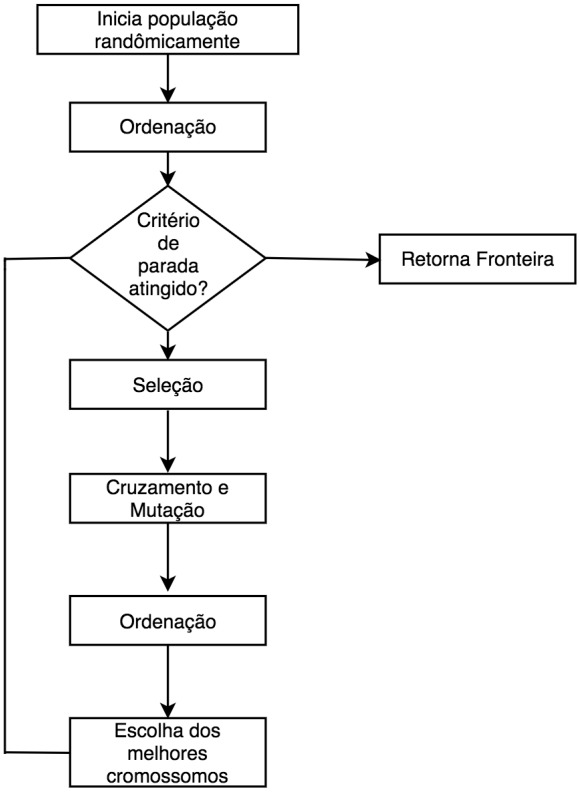
Esta seção apresenta um breve resumo do NSGA-II, uma vez que ela foi usada apenas para efeitos comparativos, e é uma meta-heurística clássica da literatura de otimização multi-

objetiva. Ela é baseada em eventos biológicos que ocorrem na genética, aonde genes (soluções) sofrem alterações através de mutações e / ou cruzamentos, onde apenas os mais adaptáveis são selecionados para perpetuar durante as gerações de indivíduos.

Sua primeira versão, a NSGA (SRINIVAS; DEB, 1994), obteve um sucesso relativo durante um longo período, até que a sua segunda versão foi proposta em Deb et al. (2002). Essa meta-heurística é utilizada na maioria das pesquisas multi-objetivas, e ainda serve de base a fundamentação de outras meta-heurísticas (COELLO; LAMONT; VELDHUIZEN, 2007), como a NCRO. Foram esses motivos levados em conta para a escolha dela como efeito comparativo nos experimentos realizados neste trabalho.

O esquema de funcionamento do NSGA-II é bastante similar ao do NCRO. Nele, a cada iteração são gerados novos indivíduos, os quais são classificados através de ranques, que medem a qualidade de suas soluções. Esses indíviduos são, então, ordenados baseados em seus ranques. A partir deles, uma nova população descendente é gerada a partir a aplicação dos operadores evolucionários de mutação e cruzamento. Assim, a meta-heurística passa a selecionar todos os indivíduos que são proprícios para a próxima geração, através do operador evolucionário de seleção. Para que a diversidade da população seja mantida, é usada a distância de *crowding*. A figura 25 apresenta um fluxograma de como o NSGA-II atua, baseado na descrição anterior.

Figura 25 – Fluxograma da execução do NSGA-II



Fonte: Próprio autor.

3
Experimentos

Esse capítulo apresenta os experimentos e elementos fundamentais que foram definidos antes do início dos mesmos. As instâncias utilizadas representam um projeto de software, englobando as propriedades apresentadas na Seção 2.3. Em cada ponto de reescalonamento, a meta-heurística utilizada retorna uma fronteira de Pareto que representa escalonamentos viáveis naquele instante. Dessa forma, um escalonamento dessa fronteira deve ser escolhido e adotado no projeto. Portanto, é descrito um seletor automatizado que determina o escalonamento que mais se aplica. A qualidade da fronteira é influenciada pelos operadores evolucionários usado e suas respectivas configurações. Para calcular a qualidade da fronteira, a métrica hipervolume foi usada. Para que o resultado obtido fosse confiável, o teste de Wilcoxon foi utilizado no conjunto de dados gerados por cada meta-heurística dos experimentos.

Os experimentos visam validar a hipótese levantada na seção 1.1 e enfatizar os objetivos desse trabalho. Os resultados estão apresentados de acordo com a convergência e a divergência das meta-heurísticas utilizadas. Assim, é ilustado a um gerente ou líder de um projeto de software como os algoritmos utilizados devem ser levados em conta, concluindo a hipótese apresentada.

3.1 Instâncias

As instâncias utilizadas nos experimentos aqui realizados foram as mesmas usadas em (SHEN et al., 2016). Elas foram criadas para a realização de testes do SPSP, em (ALBA; CHICANO, 2007), e foram adaptadas para o DSPSP, uma vez que não há *benchmarks* padrões. Dessa forma, as instâncias variam em torno dos parâmetros mais importantes do SPSP, sendo o número de funcionários, o número de tarefas e de o número de habilidades dos funcionários. A adaptação para o DSPSP adicionou parâmetros para a incerteza no custo das tarefas, os três diferentes tipos de eventos dinâmicos, o número máximo de funcionários em uma tarefa, a proficiência dos funcionários em suas habilidades e o trabalho extra realizado pelos funcionários.

Cada instância, de um total de 16[1], representa um cenário diferente para um projeto de software. A quantidade total de habilidades diferentes necessário para um projeto é de 10, em cada instância, e cada tarefa requer 5 dessas habilidades, escolhida de forma aleatória. O número de funcionários varia entre 5, 10 e 15, e o número de habilidades possuídas por eles varia entre 4 a 5, ou de 6 a 7. Vinte por cento desses funcionários trabalham de forma parcial, onde suas dedicações máximas são geradas aleatoriamente, de forma uniforme, entre $[0,5;1)$; outros 20 por cento trabalham de forma excessiva, onde suas dedicações máximas variam entre $(1;1,5]$; o restante possui dedicação máxima de $1,0$ (dia normal de trabalho).

Os funcionários saem e retornam ao projeto de acordo com uma distribuição de Poisson. Os valores escolhidos tentaram manter, em média, o funcionário disponível, para o projeto, $95,83\%$ do tempo. O salário normal dos funcionários são retirados de uma distribuição normal, com média 10.000 e desvio padrão 1000. O salário por excesso de trabalho é dado como o salário normal multiplicado por 3.

O número inicial de tarefas pode ser de 10, 20 ou 30. Ao longo do projeto, 10 novas tarefas são adicionadas ao projeto com um intervalo (em média) de 1 mês entre elas, sendo 20 por cento destas tarefas urgentes e os outros 80 por cento restantes tarefas regulares. A variância dos preços das tarefas segue uma distribuição normal. Os valores escolhidos tentaram manter, em média, 10 como sendo o preço de cada tarefa, e 5 o seu desvio padrão. O grafo de precedência é construído em cima de uma lista de adjacência, sendo que, para cada tarefa adicionada ao projeto, urgente ou regular, a mesma é inserida ao grafo precedendo ou sucedendo uma tarefa ainda não terminada, escolhida aleatoriamente, respectivamente.

Por fim, todas as instâncias são nomeadas da seguinte forma: $sT\#1\_dT\#2\_E\#3\_SK\#4-\#5$, onde $sT\#1$ significa o número inicial de tarefas, $dT\#2$ a quantidade de novas tarefas, $E\#3$ a quantidade de funcionários e $SK\#4-SK\#5$ a quantidade de habilidade que os funcionários tem, que pode ser #4 ou #5.

3.2 Seletor de escalonamentos

Para cada evento dinâmico, o projeto é atualizado e a busca por novos escalonamentos inicia-se uma outra vez. Ao final de cada busca, um conjunto de escalonamentos possíveis (a população retornada pela meta-heurística) é formado, aonde deve ser escolhido um desses para ser usado no projeto. A decisão de escolha de um escalonamento deve ser automatizada, uma vez que não é prático ter um usuário escolhendo manualmente (SHEN et al., 2016). Dessa forma, (SHEN et al., 2016) desenvolveram um método de escolha automática de um escalonamento, descrita a seguir:

Passo 1: Construção da Matriz de Comparação par-a-par. O modelo proposto para o DSPSP

[1] Shen et al. (2016) utilizou 21 devido, por exemplo, à ocorrência de novas restrições de prazo para a conclusão do projeto.

possui $n = 4$ objetivos, de tal forma que a matriz analisa a questão "Quão importante é o objetivo f_i em relação ao objetivo f_j? $(i, j = \{1, 2, ..., n\}, i < j)$". Dessa forma, existem $n \times (n-1)/2 = 6$ comparações a serem feitas. A relevância de cada uma dessas comparações é definida, então, na Matriz de Comparação par-a-par $A = (a_{ij})_{n \times n}$ através do processo de escolha analítica por hierarquia, o qual descreve o grau de preferência entre os objetivos.

Passo 2: Estimativa do vetor de pesos $\gamma = (\gamma_i)_{n \times 1}$ para os múltiplos objetivos. A média geométrica de cada linha da matriz A é calculada, a qual é normalizada a partir da divisão da soma de delas.

Passo 3: Normalização dos valores dos objetivos. Cada objetivo é normalizado de acordo com

$$n\_f_i(x) = (f_i^{max} - f_i(x))/(f_i^{max} - f_i^{min}), i = \{1, 2, ..., n\}, \tag{3.1}$$

onde f_i^{max} e f_i^{min} são, respectivamente, o valor do objetivo máximo e mínimo dentre todos os valores do conjunto de escalonamentos obtidos durante o re-escalonamento anterior.

Passo 4: Cômputo do valor da utilidade. A média geométrica ponderada dos valores dos objetivos é usada para achar o valor da utilidade de cada solução:

$$U(x) = \prod_{i=1}^{n} n\_f_i(x)^{\gamma_i / \Sigma_i^n \gamma_i} \tag{3.2}$$

Passo 5: Escolha do escalonamento com o valor máximo de utilidade como o escalonamento a ser utilizado.

Apenas os passos 3, 4 e 5 são executados a cada re-escalonamento. Dessa forma, a Matriz de Comparação par-a-par e o vetor de pesos são determinados antes mesmo do escalonamento inicial. Nos experimentos realizados para esse trabalho, foi assumido que a matriz é:

$$A = (a_{ij})_{4 \times 4} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 1/2 & 1/2 & 1 & 1 \\ 1/2 & 1/2 & 1 & 1 \end{bmatrix}. \tag{3.3}$$

Assim, o vetor de pesos correspondente é $w = (w_i)_{4 \times 1} = [0,333\ 0,333\ 0,1667\ 0,1667]^T$.

3.3 Parametrização e Operadores

Um dos passos que fazem parte dos experimentos é a definição dos operadores evolucionários e dos parâmetros a serem utilizados. Ambos possuem um impacto decisivo no desempenho da meta-heurística, e os valores utilizados nesse trabalho são valores comuns em trabalhos da literatura relacionada, o que possibilita uma comparação de igual para igual tanto

Tabela 5 – Operadores evolucionários usados em ambas as meta-heurísticas.

Operador	Valor (Probabilidade, Distribuição)
Polynomial Mutation	$1/n \times m$, 20,0
Simulated Binary Crossover	0,9, 20,0

Fonte: Próprio autor.

Tabela 6 – Valores dos parâmetros utilizados em ambas as meta-heurísticas.

Parâmetro	Valor
Tamanho da População	100 indivíduos
Máximo de avaliações de objetivo (critério de parada)	10.000 avaliações

Fonte: Próprio autor.

entre as meta-heurísticas usadsa neste trabalho, quanto para investigações futuras. Eles estão descritos descritos nas tabelas 5 e 6.

Os operadores *Polynomial Mutation* e *Simulated Binary Crossover* são comuns às duas meta-heurísticas, e tratam, respectivamente, da mutação dos genes (no NSGA-II) e na geração de novas moléculas (no NCRO) a partir da reação de decomposição. A **Reação 1** e a **Reação 2** também utilizam um operador de mutação, porém com um efeito menor do que o apresentado anteriormente. O operador de *crossover* é responsável, respectivamente, pelo cruzamento dos genes (no NSGA-II) e por unir duas moléculas (no NCRO).

Além destes, o NSGA-II também possui um operador exclusivo para a seleção (*selection*) de genes nos quais serão aplicados os passos evolucionários (cruzamento e mutação). Ele funciona escolhendo dois genes da população de acordo com atributos do mesmo (por exemplo, restrições violadas por ele). Existem outros parâmetros para o NCRO, definidos na forma em que são apresentados na Tabela 7.

Tabela 7 – Parâmetros de execução do NCRO.

Parâmetro	Valor
Energia inicial	0,0
Energia Cinética inicial	10.000
Seletor molecular	0,5
Taxa de perda de Energia Cinética	0,2
Step (Mutação)	0,12
Limiar da Síntese	20,0
Limiar da Decomposição	15,0

Fonte: Próprio autor.

Em ordem, a energia inicial define o quanto de energia o *buffer* irá conter ao início da

execução. A energia cinética inicial define a quantidade de KE que cada molécula irá conter inicialmente. O parâmetro de escolha entre as reações define quando uma reação será unimolecular ou multicelular. A taxa de perda de energia cinética que uma molécula sofre a cada reação é também definida. O parâmetro *step* colabora com a cota da mutação feita nas moléculas que sofrem a **Reação 1** e a **Reação 2**. Por fim, os parâmetros limiares da síntese e decomposição determinam quando essas reações podem ser realizadas, detalhado na seção 2.7.1. Por último, em cada execução independente o tamanho da população foi de 100 indivíduos e processo de busca se encerra após 10.000 avaliações de objetivo.

A escolha dos parâmetros e seus valores deu-se da seguinte forma: os operadores da Tabela 5 são operadores padrões em testes de otimização, assim como os seus valores. Assim, foi optado por utilizá-los nessa configuração sem que fossem alterados. A mesma lógica segue para os parâmetros do NSGA-II. Contudo, para o NCRO (Tabela 7) a escolha foi determinada de forma empírica. Para cada parâmetro, um intervalo de valor foi definido e algum valor contido no intervalo era selecionado de forma aleatória, onde o modelo era executado em seguida para que o desempenho da metaheurística fosse verificado. Nesse intuito, o intervalo buscava convergir para algum valor que maximizava o desempenho do NCRO.

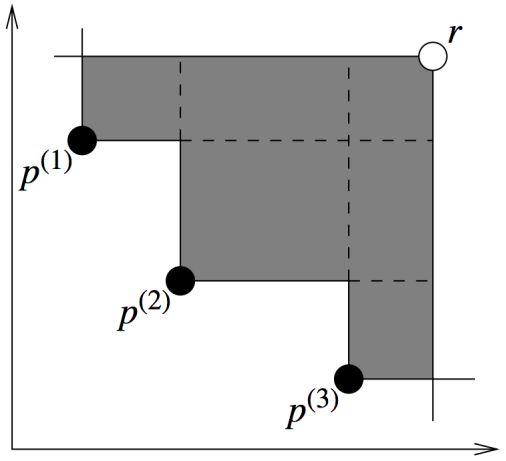
Os valores da tabela 6 também foram definidos pela metodologia de tentativa e erro. De fato, assim como os outros parâmetros citados anteriormente, tentativa e erro é a metodologia mais utilizada para a definição e escolhas de parâmetros e valores (SABAR et al., 2013). O critério de parada é um fator crucial para a qualidade dos resultados. Um valor maior para a avaliações de funções objetivos aumenta as chances da meta-heurística retornar soluções melhores. Por outro lado, o tempo de execução pode aumentar muito e sem proporções, prejudicando a eficiência em relação ao tempo disponível para os testes.

3.4 Métricas de Comparação de Resultados

3.4.1 Hipervolume

Os resultados desse trabalho serão avaliados pela métrica Hipervolume (BEUME et al., 2009). Essa é uma das métricas mais usada em trabalhos de otimização com meta-heurísticas evolucionárias, uma vez que considera diversos aspectos do resultado obtido, como a diversidade da fronteira e a sua convergência. Para computar o hipervolume, após a geração da fronteira de Pareto, é escolhido um ponto de *referência*, que é um ponto não alcançado por nenhuma das soluções. Em problemas de minimização, o ponto de referência é um ponto ruim. Em seguida, é calculada a área do polígono formado a partir do ligamento de todas as soluções que compõem a fronteira e o ponto de referência. Assim, quanto maior a área do polígono, melhor. A Figura 26 ilustra esse processo.

Figura 26 – Hipervolume de uma Fronteira de Pareto de duas dimensões. O ponto de referência é representado por r, enquanto que os pontos da Fronteira por $p^{(1)}, p^{(2)}, p^{(3)}$.



Fonte: Adaptado de (FONSECA; PAQUETE; LÓPEZ-IBÁÑEZ, 2006)

3.4.2 Teste de Wilcoxon

O teste de Wilcoxon é um teste estatístico não-paramétrico usado para testar hipóteses levantadas sobre um determinado evento. Dadas duas amostras observadas, o teste de Wilcoxon computa a probabilidade, chamada de *p-value*, das duas populações serem diferentes por chances do acaso. Para isso, o resultado é determinado de acordo com um nível de significância, que remete à probabilidade de rejeitar uma hipótese nula quando a mesma é verdadeira. Valores comuns na literatura para o nível de significância são $0,05$ e $0,01$.

3.5 Questões de Pesquisa

As questões de pesquisas que serão investigadas a partir dos resultados gerados são:

QP1: Qual meta-heurística obteve melhores resultados, de acordo com a métrica hipervolume?

QP2: Por que há momentos, em certas instâncias usadas nos experimentos, que o valor do hipervolume cai de forma abrupta?

Se tratando de heurísticas, as quais não oferecem garantias de resultados otimais (e tão pouco iguais), para uma mesma instância e algoritmo executado, o resultado obtido pode ser diferente. Dessa forma, é reforçado a necessidade de uso do Teste de Wilcoxon. Para aumentar o nível de confiança dos resultados, cada instância foi executada 10 vezes, onde foi coletada a fronteira de Pareto de cada evento de cada execução, que servirá como parte da amostra. Por fim, os testes estatísticos foram realizados a um nível de $0,05$ de significância.

3.5.1 QP1: Comparação das Meta-heurísticas com o Hipervolume

Um projeto é composto de vários eventos dinâmicos que ocorrem ao longo de sua execução. Assim, para cada execução do projeto (em cada algoritmo), foi coletada a fronteira de Pareto encontrada pela meta-heurística, sendo posta em um arquivo único de cada evento. Ao final de todas as execuções, cada evento do projeto possuía um arquivo com todas as Fronteiras encontradas. Essa, por sua vez, foi normalizada (de acordo com a Equação 3.1) para que houvesse uma comparação escalável entre os hipervolumes obtidos.

Como os valores de cada objetivo variavam entre $[0;1]$, o ponto de refência usado para o cômputo do hipervolume foi $(1,1,1,1,1,1)$ para o escalonamento inicial, e $(1,1,1,1,1,1,1,1)$ para os demais. Dessa forma, gerou-se uma amostra de 10 hipervolumes, os quais foram usados no Teste de Wilcoxon, com hipóteses bilaterais. A hipótese nula H_0 foi postulada como sendo: não há diferença entre o conjunto de dados gerados nos experimentos. Já a hipótese alternativa H_1, por sua vez, afirma que o conjunto de dados possui diferença estatística, indicando que um algoritmo foi melhor que o outro naquela instância.

A tabela 8 ilustra os valores obtidos pelo teste de Wilcoxon em cada instância. Nela, a coluna do *p-value* está especificada seguida do veredito final (aceita ou recusa a hipótese nula). Por conta de problemas de precisões, os resultados apresentados são aproximados.

Tabela 8 – Resultados obtidos pelo Teste de Wilcoxon para cada uma das 16 instâncias. **N** para **não** rejeitar a hipótese, **R** para **rejeitar** a hipótese nula.

Instância	*p-value*	Veredito
$sT10\_dT10\_E5\_SK4-5$	$= 7,503 \times 10^{-5}$	**R**
$sT10\_dT10\_E10\_SK4-5$	$= 3,284 \times 10^{-2}$	**R**
$sT10\_dT10\_E15\_SK4-5$	$= 1,487 \times 10^{-6}$	**R**
$sT10\_dT10\_E5\_SK6-7$	$= 5,603 \times 10^{-10}$	**R**
$sT10\_dT10\_E10\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**
$sT10\_dT10\_E15\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**
$sT20\_dT10\_E5\_SK4-5$	$= 1,014 \times 10^{-13}$	**R**
$sT20\_dT10\_E10\_SK4-5$	$< 2,2 \times 10^{-16}$	**R**
$sT20\_dT10\_E15\_SK4-5$	$< 2,2 \times 10^{-16}$	**R**
$sT20\_dT10\_E5\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**
$sT20\_dT10\_E10\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**
$sT20\_dT10\_E15\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**
$sT30\_dT10\_E5\_SK4-5$	$< 2,2 \times 10^{-16}$	**R**
$sT30\_dT10\_E10\_SK4-5$	$< 2,2 \times 10^{-16}$	**R**
$sT30\_dT10\_E15\_SK4-5$	$< 2,2 \times 10^{-16}$	**R**
$sT30\_dT10\_E5\_SK6-7$	$< 2,2 \times 10^{-16}$	**R**

Fonte: Próprio autor.

De acordo com os testes, as meta-heurísticas utilizadas geraram um conjunto de dados com diferenças estatísticas. O gráfico dos hipervolumes obtidos na instância 12 (figura 27), por

exemplo, mostra que o NSGA-II foi superior ao NCRO na maior parte da execução do projeto. A tabela 9 mostra que, de fato, o NSGA-II obteve hipervolumes melhores do que o NCRO em 13 das 16 instâncias. Ainda, é possível observar que, através dos valores computados pelo desvio-padrão, o NSGA-II foi mais estável do que o NCRO em 11 das 16 instâncias, no sentido de que o primeiro gerou hipervolumes que não variaram muito ao longo do projeto, enquanto que o segundo obteve hipervolumes mais esparsos.

Tabela 9 – Médias e Desvios-padrão dos hipervolumes gerados por cada algoritmo em cada instância.

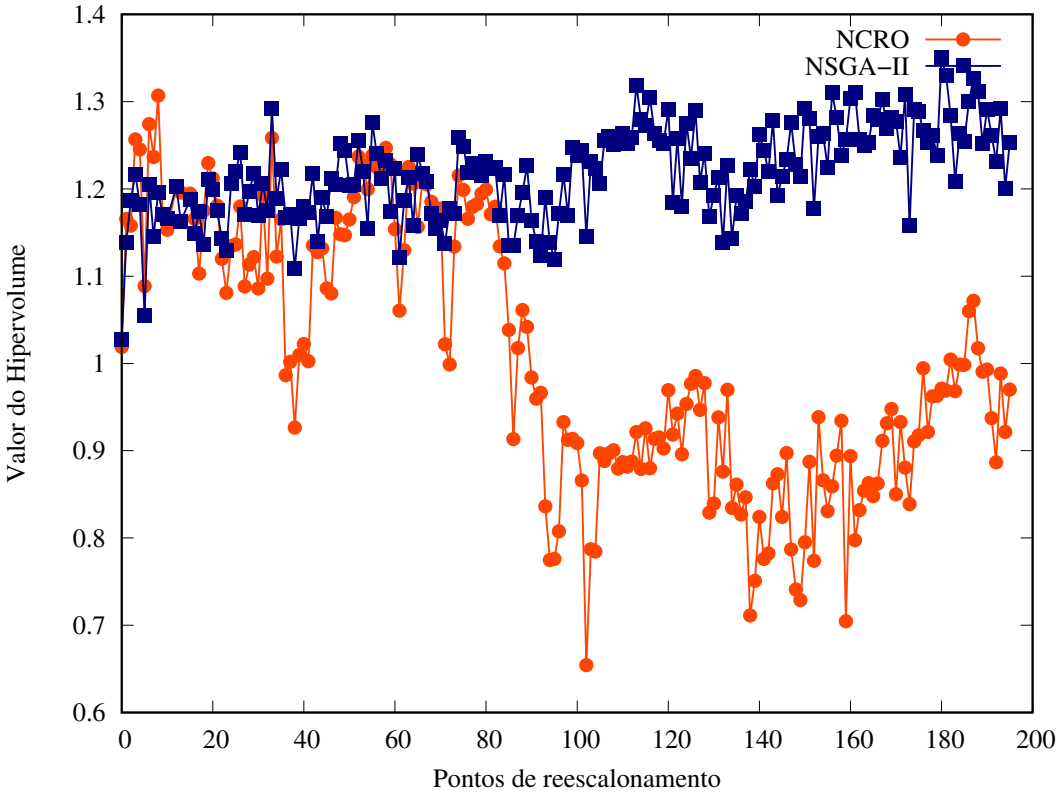
Instância	NCRO		NSGA-II	
	Média	Desvio	Média	Desvio
$sT10\_dT10\_E5\_SK4-5$	$6,774 \times 10^{-1}$	$4,32 \times 10^{-1}$	$6,557 \times 10^{-1}$	$4,22 \times 10^{-1}$
$sT10\_dT10\_E10\_SK4-5$	$7,815 \times 10^{-1}$	$1,524 \times 10^{-1}$	$7,421 \times 10^{-1}$	$2,339 \times 10^{-1}$
$sT10\_dT10\_E15\_SK4-5$	$9,260 \times 10^{-1}$	$1,576 \times 10^{-1}$	$8,818 \times 10^{-1}$	$1,339 \times 10^{-1}$
$sT10\_dT10\_E5\_SK6-7$	$8,896 \times 10^{-1}$	$3,970 \times 10^{-1}$	$9,251 \times 10^{-1}$	$4,197 \times 10^{-1}$
$sT10\_dT10\_E10\_SK6-7$	$10,491 \times 10^{-1}$	$1,057 \times 10^{-1}$	$11,330 \times 10^{-1}$	$0,893 \times 10^{-1}$
$sT10\_dT10\_E15\_SK6-7$	$9,918 \times 10^{-1}$	$1,021 \times 10^{-1}$	$10,372 \times 10^{-1}$	$0,982 \times 10^{-1}$
$sT20\_dT10\_E5\_SK4-5$	$9,380 \times 10^{-1}$	$2,640 \times 10^{-1}$	$11,155 \times 10^{-1}$	$2,710 \times 10^{-1}$
$sT20\_dT10\_E10\_SK4-5$	$9,271 \times 10^{-1}$	$0,963 \times 10^{-1}$	$10,424 \times 10^{-1}$	$1,156 \times 10^{-1}$
$sT20\_dT10\_E15\_SK4-5$	$10,221 \times 10^{-1}$	$1,223 \times 10^{-1}$	$11,420 \times 10^{-1}$	$0,852 \times 10^{-1}$
$sT20\_dT10\_E5\_SK6-7$	$9,198 \times 10^{-1}$	$1,922 \times 10^{-1}$	$10,772 \times 10^{-1}$	$1,596 \times 10^{-1}$
$sT20\_dT10\_E10\_SK6-7$	$8,930 \times 10^{-1}$	$1,601 \times 10^{-1}$	$9,630 \times 10^{-1}$	$1,587 \times 10^{-1}$
$sT20\_dT10\_E15\_SK6-7$	$10,082 \times 10^{-1}$	$1,500 \times 10^{-1}$	$12,180 \times 10^{-1}$	$0,534 \times 10^{-1}$
$sT30\_dT10\_E5\_SK4-5$	$8,497 \times 10^{-1}$	$2,786 \times 10^{-1}$	$12,138 \times 10^{-1}$	$2,978 \times 10^{-1}$
$sT30\_dT10\_E10\_SK4-5$	$8,590 \times 10^{-1}$	$1,809 \times 10^{-1}$	$12,296 \times 10^{-1}$	$0,843 \times 10^{-1}$
$sT30\_dT10\_E15\_SK4-5$	$9,450 \times 10^{-1}$	$1,506 \times 10^{-1}$	$11,539 \times 10^{-1}$	$1,408 \times 10^{-1}$
$sT30\_dT10\_E5\_SK6-7$	$10,008 \times 10^{-1}$	$2,400 \times 10^{-1}$	$12,200 \times 10^{-1}$	$1,682 \times 10^{-1}$

Fonte: Próprio autor.

Ainda tomando como exemplo a instância 12, à medida que o projeto foi executado, o hipervolume do NCRO decresce de forma notória, enquanto que o do NSGA-II se manteve estável. Na altura do Ponto de Reescalonamento 110, por exemplo, é possível verificar (Tabela 10) que o NSGA-II obteve melhores objetivos do que o NCRO. Os valores expressados pela Tabela 10 foram calculados de forma similar do hipervolume: na instância 12, foram juntadas, em um arquivo para cada algoritmo, todas as 10 fronteiras de Pareto (relativas ao evento 110) geradas pelo NCRO e do NSGA-II. Assim, foi possível calcular as medidas estatísticas referentes a uma população, como média e desvio padrão (entre parênteses). Observa-se que o NSGA-II, com exceção do objetivo de estabilidade, obteve melhores médias e desvios padrões, o que reflete os valores dos hipervolumes computados, e corrobora com a estabilidade do mesmo.

Uma possível explicação para essa diferença entre os hipervolumes entre os algoritmos pode ser encontrada na quantidade e na configuração dos respectivos parâmetros das meta-heurísticas. Enquanto que, além dos dois operadores comuns ao NCRO e ao NSGA-II, descritos

Figura 27 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 12).



Fonte: Próprio autor.

Tabela 10 – Média e desvio padrão (entre parênteses) dos 4 objetivos NCRO e do NSGA-II na instância 12, no evento 110.

	Duração	Custo	Robustez	Estabilidade
NCRO	$128,27 \times 10^5 (126,36 \times 10^5)$	$4282,68 \times 10^5 (424,17 \times 10^5)$	$56,88(56,17)$	$3407,94 \times 10^5 (33,82 \times 10^5)$
NSGA-II	$790,51 \times 10^5 (756,80 \times 10^5)$	$2066,49 \times 10^5 (204,37 \times 10^5)$	$47,78(47,28)$	$6343,01 \times 10^5 (62,61 \times 10^5)$

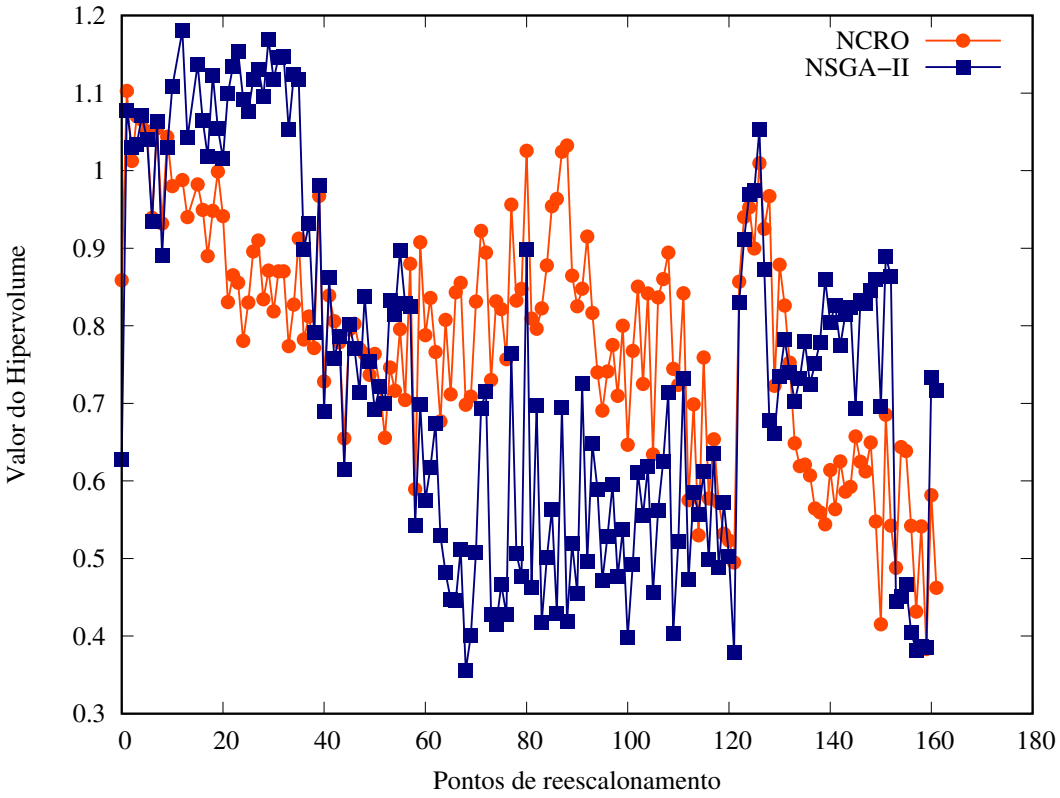
Fonte: Próprio autor.

na tabela 5, o primeiro possui outros 7 parâmetros determinados antes do início da execução, contra apenas 1 do segundo. Uma grande quantidade de pârametros torna o processo de escolha mais difícil, uma vez que a escolha costuma dar-se de forma empírica, sem qualquer formalidade descrita. E, sendo que os parâmetros aplicam uma grande influência no desempenho da meta-heurística, uma má escolha pode acarretar em interpretações erradas sobre o desempenho daquele algoritmo em um determinado problema.

O NCRO mostrou-se como um algoritmo melhor nas três menores instâncias utilizadas nos experimentos. O gráfico da figura 28, em particular, ilustra o seu desempenho ao longo do projeto, de acordo com a média dos hipervolumes em cada evento. É possível notar que, apesar do início inferior ao do NSGA-II, o NCRO começa a gerar soluções mais competitivas à medida que o projeto foi executado. Logo, pode-se dizer que, com essas configurações dos parâmetros,

é recomendável o uso do NCRO para a geração de novos escalonamentos em projetos que possuem poucas tarefas e poucos funcionários. Acredita-se que a configuração dos parâmetros possibilitou ao NCRO fazer uma varredura melhor no espaço de busca local. Em contraste, o NSGA-II apresenta-se como um algoritmo a ser utilizado em projetos maiores, sobretudo quando a estabilidade da geração de escalonamentos deve ser levada em conta, caso de um projeto de software, onde preza-se sempre por um desempenho bom e regular.

Figura 28 – Gráfico dos Hipervolume à medida que o projeto era executado (instância 2).



Fonte: Próprio autor.

3.5.2 QP2: A queda abrupta do valor do Hipervolume em certos instantes

Em algumas instâncias, é possível observar a queda abrupta do hipervolume em alguns instantes. Existem duas causas para que isso tenha acontecido. Os motivos base estão relacionados ao evento que ocorreu e a maneira como foi os objetivos foram computados. Na instância 1, no evento 4, o funcionário 2 saiu do projeto, causando a violação da restrição **Restrição 2**. Nesse caso, os objetivos são penalizados como especificado na Seção 2.3.7.2.

A escolha manual dos objetivos acabou prejudicando o motor de busca da meta-heurística. De fato, apesar do tamanho máximo da população final ser 100, tanto o NCRO quanto o NSGA-II conseguiram encotrar apenas 10 pontos não dominados. Especificamente na instância 1, no evento 4, o NSGA-II encontrou os pontos não dominados descritos na tabela 11. Como

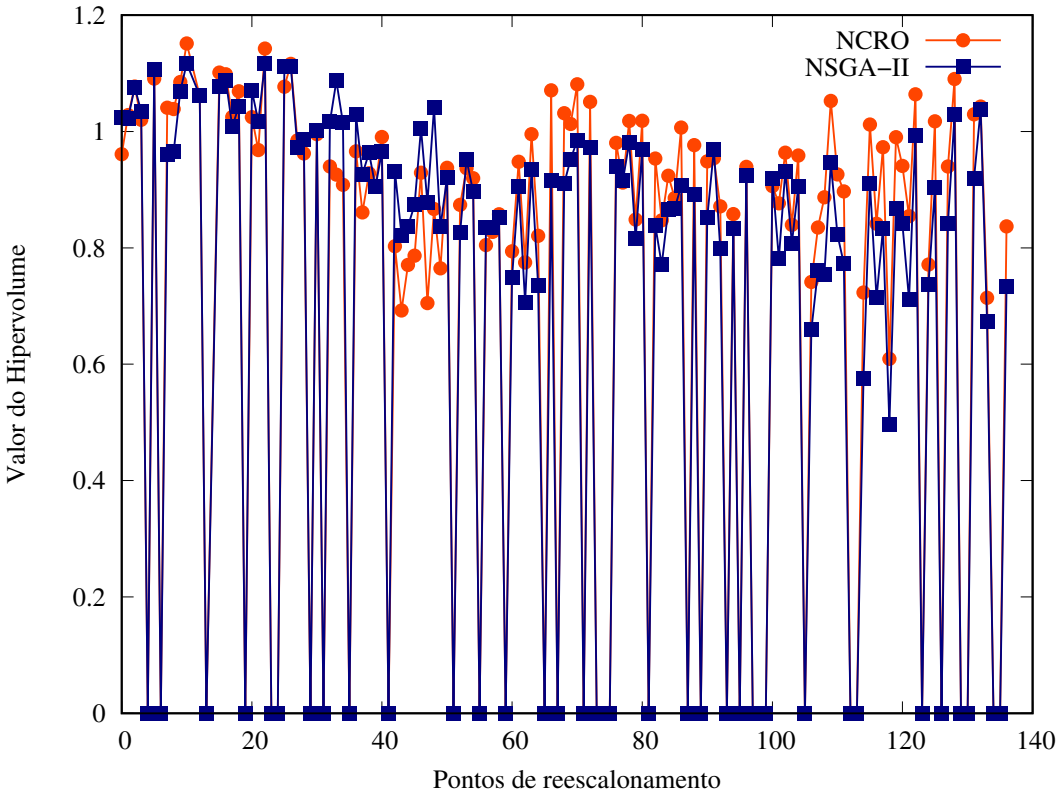
Tabela 11 – Pontos (aproximados) não dominados encontrados pelo NSGA-II com a violação de habilidades violada (instância 1, evento 4).

Ponto	Duração	Custo	Robustez	Estabilidade
1	$4044054,98$	174843×10^8	$4,0$	$288,0$
2	$4043656,83$	174825×10^8	$4,0$	$288,0$
3	$4041042,15$	174712×10^8	$4,0$	$288,0$
4	$4037958,59$	174579×10^8	$4,0$	$288,0$
5	$4041665,64$	174739×10^8	$4,0$	$288,0$
6	$4039901,86$	174663×10^8	$4,0$	$288,0$
7	$4041829,88$	174746×10^8	$4,0$	$288,0$
8	$4041580,35$	174736×10^8	$4,0$	$288,0$
9	$4038725,96$	174612×10^8	$4,0$	$288,0$
10	$4040166,96$	174674×10^8	$4,0$	$288,0$

Fonte: Próprio autor.

consequência dos valores da robustez serem iguais, não foi possível normalizar as soluções de acordo com a Equação 3.1, pois a mesma resulta em uma divisão por 0. Dessa forma, foi optado por representar o hipervolume como sendo $0,0$. O gráfico da média dos hipervolumes gerados por evento ilustra essa anomalia, na figura 45.

Figura 29 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 1).



Fonte: Próprio autor.

Essa anomalia acontece em projetos que possuem poucos funcionários ou quando um funcionário, sozinho, detém muitas habilidades exclusivas. A sua eventual saída pode ocasionar a violação em alguma tarefa que possui alguma habilidade específica, e que nesse caso não haveria funcionário disponível para a execução da mesma. No exemplo da tabela 11, a tarefa 2 requer a habilidade 5 para ser executada. Como apenas o empregado 4 possui, não é possível executa-lá. No Apêndice B, essa anomalidade é usualmente encontrada nas instâncias as quais a quantidade total de funcionários no projeto é 5 (instâncias 1, 4, 7, 10, 13 e 16).

Uma possível solução para esse problema é redefinir a penalidade dos objetivos. Na literatura, existem estudos e técnicas de penalização de objetivos (ABRAHAM; JAIN, 2005). Uma técnica diferente que poderia ser aplicada seria alterar o modelo do DSPSP proposto por (SHEN et al., 2016), para permitir que os objetivos sejam calculados mesmo violando a restrição. Ao final, seriam aplicados valores grandes o suficiente para que nenhuma solução possível alcançasse esses objetivos penalizados. Assim, à medida que as meta-heurísticas explorassem o espaço de busca, essas soluções seriam dominadas facilmente, sendo eventualmente excluídas da população.

3.6 Ambiente utilizado

Para a implementação do modelo do DSPSP por Shen et al. (2016), foi utilizado o *framework* jMetal[2]. Este ambiente foi desenvolvido na linguagem de programação Java[3] e tem como objetivo fornecer ferramentas e algoritmos implementados relacionados à área de otimização multiobjetiva. Dessa forma, a meta-heurística NSGA-II já estava implementada. A versão utilizada do jMetal foi a versão 5.3.

O NCRO foi adaptado para que fosse possível utilizar o modelo do DSPSP, implementado de acordo com a arquitetura do jMetal, em seu código fonte (disponibilizado pelos próprios autores[4]). Os gráficos gerados a partir dos hipervolumes médios e dos pontos de reescalonamento foram construídos com a ferramenta GNUPLOT[5].

Os testes estatísticos (média, desvio padrão, teste de Wilcoxon) foram executados dentro do ambiente da linguagem R[6]. Por fim, as execuções do modelo foram realizadas em dois computadores: MacBook Pro mid 2015 com 16GB de RAM, Intel i7 2,2GHz, equipado com macOS High Sierra, e um segundo, Sony Vaio com 4GB de RAM, Intel i5 1,8GHz, equipado o SO Arch Linux.

[2] Disponível em: <http://jmetal.sourceforge.net/>.
[3] Disponível em: <https://www.java.com>.
[4] Disponível em: <https://sites.google.com/site/slimbechikh/sourcecodes>.
[5] Disponível em: <http://www.gnuplot.info/>.
[6] Disponível em: <www.r-project.org>

4

Conclusões

Este trabalho investigou a versão dinâmica do Problema do Escalonamento em Projeto de Software. O modelo escolhido foi proposto recentemente por (SHEN et al., 2016), e incorpora diversas características de um projeto de software da vida real: a chegada de tarefas, idas e vindas de funcionários, escalonamento proativo, entre outras.

Ao adotar a SBSE como metodologia de investigação, aplicou-se as meta-heurísticas, capazes de buscarem e otimizarem soluções de problemas que possuem um grande espaço de busca de soluções, como o caso do DSPSP. A necessidade de investigar um problema com diferentes meta-heurísticas é justificada no teorema *No Free Lunch*, que afirma que é impossível uma meta-heurística ser boa em todos os problemas, e que apenas testando diferentes meta-heurísticas pode-se obter bons resultados para um determinado problema.

Sendo assim, foram escolhidos os algoritmos *Nondominated Sorting Genetic Algorithm II* e o *Nondominated Sorting Chemical Reaction Optimization*. O primeiro, NSGA-II, é um algoritmo clássico da literatura de otimização, que possui bons resultados em vários problemas. O segundo, recém proposto, apresenta resultados melhores em alguns problemas da literatura, quando comparado ao NSGA-II, como na versão estática do SPSP, e por isso justifica a investigação deste trabalho.

A implementação do modelo foi realizada assim que o mesmo foi decidido. O código está disponível online[1] e para livre uso. Esse também acompanha o código adaptado do NCRO. O passo seguinte, e que requisitou um maior tempo de trabalho, foi a execução das instâncias. Essa fase demandou horas de execução até que todas as 10 execuções das 16 instâncias fossem terminadas. Para a instância 16, por exemplo, cerca de 26 horas foram necessárias para executá-la no NSGA-II, contra 30 horas de processamento da mesma instância no NCRO. A análise dos resultados foi realizada ao término dos experimentos, após a geração dos valores do hipervolume das fronteiras e dos respectivos gráficos, concluindo, portanto, todos os objetivos específicos

[1] Disponível em: <https://git.dcomp.ufs.br/jjaneto/ncro>

levantados.

Os experimentos mostraram que, ao menos quando a quantidade de tarefas em um projeto cresce, o NCRO não consegue manter a sua eficiência, e obtem resultados inferiores ao do NSGA-II. Este, por sua vez, mantém-se estável, gerando resultados que não variam muito, o que mostra a qualidade do algoritmo em relação a escalabilidade. Um dos possíveis problemas para o qual o NCRO não conseguiu manter o desempenho pode ser a escolha dos parâmetros, na qual é fundamental para permitir o algoritmo a convergir e divergir no espaço de busca de um problema.

Portanto, um possível trabalho futuro é estudar de forma mais profunda os parâmetros do algoritmo NCRO, em especial, aplicado ao DSPSP. Encontrar um balanço entre todos (num total de 7) é um estudo denso, que demanda testes e uma literatura relacionada ainda pouco explorada. Trabalhos futuros irão explorar os efeitos de alterar a importância dos objetivos no seletor de escalonamento, bem como a implementação de algum método que possa ser utilizado para reinicializar as matrizes de dedicações entre os eventos, procurando otimizar os valores dos objetivos.

Em relação ao modelo, existem diversas linhas de estudos que valem o trabalho. Uma forma de adaptação à vida real poderia levar em conta como o excesso de trabalho é tratado. Além do limite máximo, o excesso poderia gerar uma queda de rendimento do projeto de forma geral à medida que o(s) funcionário(s) estivesse(m) sujeito(s) a situação. Outro ponto de estudo poderia adaptar os escalonamentos de acordo com o histórico dos funcionários em projetos ou tarefas anteriores. Nesse caso, seria priorizada uma equipe que maximiza a produção de uma determinada tarefa considerando a performance desta equipe em projetos similares anteriores.

Referências

ABRAHAM, A.; JAIN, L. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. [S.l.]: Springer, 2005. (Advanced Information and Knowledge Processing). Citado na página 61.

ALBA, E.; CHICANO, F. Software project management with gas. *Information Sciences*, Elsevier, v. 177, p. 2380–2401, 2007. Citado 4 vezes nas páginas 15, 16, 23 e 50.

ANDRADE, J. J. N.; SILVA, L.; CARVALHO, A. B. Explorando o problema de escalonamento de projeto com meta-heurísticas. In: *Anais do Workshop de Trabalhos de Iniciação Científica e Graduação (WTICG)*. [S.l.: s.n.], 2017. p. 143–152. Citado 2 vezes nas páginas 15 e 16.

ANTONIOL, G.; PENTA, M. D.; HARMAN, M. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In: IEEE. *Software Metrics, 2004. Proceedings. 10th International Symposium on*. [S.l.], 2004. p. 172–183. Citado na página 25.

BECHIKH, S.; CHAABANI, A.; SAID, L. B. An efficient chemical reaction optimization algorithm for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 45, n. 10, p. 2051–2064, 2015. Citado 4 vezes nas páginas 16, 41, 44 e 46.

BEUME, N. et al. On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 13, n. 5, p. 1075–1082, 2009. Citado na página 54.

CARVALHO, M. H. de et al. *Uma Introdução Sucinta a Algoritmos de Aproximação*. [S.l.: s.n.], 2001. Citado na página 14.

CHANG, C. K. et al. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, Elsevier, v. 50, n. 11, p. 1142–1154, 2008. Citado 2 vezes nas páginas 33 e 34.

CHEN, W.; ZHANG, J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, IEEE, v. 39, p. 1–17, March 2013. Citado 2 vezes nas páginas 15 e 21.

CHICANO, F. et al. A novel multiobjective formulation of the robust software project scheduling problem. In: SPRINGER. *Applications of Evolutionary Computation*. [S.l.]: Springer-Verlag Berlin, 2012. p. 497–507. Citado na página 25.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2rd. ed. Nova Iorque: Springer, 2007. Citado 6 vezes nas páginas 15, 19, 20, 38, 40 e 49.

CRAWFORD, B. et al. Intelligent water drop algorithm (iwd) to solve software project scheduling problem. In: IEEE. *Information Systems and Technologies (CISTI), 2016 11th Iberian Conference on*. [S.l.], 2016. p. 1–4. Citado na página 15.

CRAWFORD, B. et al. A max–min ant system algorithm to solve the software project scheduling problem. *Expert Systems with Applications*, Elsevier, v. 41, p. 6634–6645, 2014. Citado 2 vezes nas páginas 16 e 38.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 2, p. 182–197, Agosto 2002. Citado 4 vezes nas páginas 16, 41, 46 e 49.

DORIGO, M. *Optimization, Learning and Natural Algorithms*. Tese (Doutorado) — Politecnico di Milano, Itália, 1992. Citado na página 15.

FERRUCCI, F.; HARMAN, M.; SARRO, F. Search-based software project management. In: RUHE, G.; WOHLIN, C. (Ed.). *Software Project Management in a Changing World*. [S.l.]: Springer Berlin Heidelberg, 2014. p. 373–399. ISBN 978-3-642-55034-8. Citado 2 vezes nas páginas 16 e 21.

FONSECA, C. M.; PAQUETE, L.; LÓPEZ-IBÁÑEZ, M. An improved dimension-sweep algorithm for the hypervolume indicator. In: *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*. Piscataway, NJ: IEEE Press, 2006. p. 1157–1163. Citado na página 55.

GLOVER, F. Tabu search—part i. *ORSA Journal on Computing*, p. 190–206, 1989. Citado na página 15.

GOLDBARG, M. R.; GOLDBARG, E. G.; LUNA, H. P. L. *Otimização combinatória e meta-heurísticas: algoritmos e aplicações*. 1th. ed. Rio de Janeiro: Elsevier, 2016. Citado 3 vezes nas páginas 20, 38 e 40.

GROUP, T. S. *CHAOS*. [S.l.], 2015. Citado 3 vezes nas páginas 15, 21 e 24.

GUEORGUIEV, S.; HARMAN, M.; ANTONIOL, G. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. [S.l.]: ACM New York, 2009. p. 1673–1680. Citado na página 25.

HARMAN, M. The current state and future of search based software engineering. In: *Future of Software Engineering*. DC: IEE Computer Society, 2007. p. 342–357. Citado na página 15.

HARMAN, M. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, v. 45, n. 1, November 2012. Citado 2 vezes nas páginas 37 e 40.

HARMAN, M.; JONES, B. F. Search-based software engineering. *Applied Soft Computing*, Elsevier, v. 43, n. 14, p. 833–839, 2001. Citado 2 vezes nas páginas 14 e 37.

HO, Y. C.; PEPYNE, D. L. Simple explanation of the no-free-lunch theorem and its implications. In: *Journal of Optimization Theory and Applications*. [S.l.]: Kluwer Academic Publishers-Plenum Publishers, 2002. v. 115, p. 549–570. Citado 2 vezes nas páginas 16 e 39.

KITCHENHAM, B. *Procedures for Performing Systematic Reviews*. [S.l.], 2004. Citado 2 vezes nas páginas 17 e 69.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado 2 vezes nas páginas 17 e 69.

KURADA, R. R.; PAVAN, K.; RAO, D. A. D. A preliminary survey on optimized multiobjective metaheuristic methods for data clustering using evolutionary approaches. v. 5, 12 2013. Citado na página 39.

LAM, A. Y. S.; LI, V. O. K. Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 14, n. 3, p. 381–399, 2010. Citado 4 vezes nas páginas 38, 39, 41 e 44.

LAM, A. Y. S.; LI, V. O. K. *Chemical Reaction Inspired Metaheuristic for Optimization*. 2010b? Disponível em: <http://homepages.dcc.ufmg.br/~cascini/cascini_paper_SBSE.pdf>. Citado 3 vezes nas páginas 41, 43 e 44.

LUNA, F. et al. The software project scheduling problem: A scability analysis of multi-objective metaheuristics. *Applied Soft Computing*, v. 15, p. 136–148, 2013. Citado na página 19.

MINKU, L. L.; SUDHOLT, D.; YAO, X. Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis. *IEEE Transactions on Software Engineering*, IEEE, v. 40, p. 83–102, October 2014. Citado na página 15.

MITCHELL, M. *An Introduction to Genetic Algorithms*. [S.l.]: MIT Press, 1998. (Complex Adaptive Systems). Citado na página 15.

OUELHADJ, D.; PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, Springer, 2008. Citado na página 24.

PEIXOTO, D. C. C.; MATEUS, G. R.; RESENDE, R. F. *Evaluation on the Search-Based Optimization Techniques to Schedule and Staff Software Projects: a Systematic Literature Review*. 2014. Disponível em: <http://homepages.dcc.ufmg.br/~cascini/cascini_paper_SBSE.pdf>. Citado na página 21.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 8th. ed. Nova Iorque: McGraw-Hill Education, 2014. Citado 5 vezes nas páginas 14, 15, 16, 20 e 21.

ROTHLAUF, F. *Design of Modern Heuristics: Principles and Application*. [S.l.]: Springer Berlin Heidelberg, 2011. (Natural Computing Series). Citado na página 14.

SABAR, N. R. et al. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, v. 17, n. 6, p. 840–661, Setembro 2013. Citado na página 54.

SHEN, X. et al. Dynamic software project scheduling through a proactive-rescheduling method. *IEEE Transactions on Software Engineering*, IEEE, v. 42, n. 7, July 2016. Citado 13 vezes nas páginas 16, 17, 23, 24, 26, 30, 32, 37, 50, 51, 61, 62 e 70.

SOMMERVILLE, I. *Software Engineeering*. 8th. ed. [S.l.]: Addison Wesley, 2006. Citado na página 20.

SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, v. 2, n. 3, p. 221–248, Setembro 1994. Citado na página 49.

WIKIPÉDIA. *Pareto efficiency — Wikipedia, The Free Encyclopedia*. 2018. <https://en.wikipedia.org/w/index.php?title=Pareto_efficiency&oldid=833551318>. Citado na página 21.

WOLPER, D. H.; MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, IEE, v. 1, p. 67–82, April 1997. Citado na página 38.

XIAO, J. et al. Dynamic resource scheduling in disruption-prone software development environments. In: *Fundamental Approaches to Software Engineering*. [S.l.]: Springer-Verlag Berlin, 2010. p. 107–122. Citado na página 25.

Apêndices

APÊNDICE A – Revisão Sistemática

Todos os passos seguiram a revisão sistemática proposta em (KITCHENHAM, 2004) e (KITCHENHAM; CHARTERS, 2007). Segundo (KITCHENHAM; CHARTERS, 2007), revisões sistemáticas buscam apresentar uma fiel avaliação acerca do campo de pesquisa seguindo rigorosos e confiáveis métodos de busca de literatura. O protocolo de revisão adotado foi dividido em 4 sub-divisões da seguinte maneira: definição do tema a ser pesquisado, escolha das fontes de pesquisas, palavras chaves e *strings* de busca, e seleção dos estudos previamente escolhidos.

A.1 Definição do tema

O objetivo principal desse TCC é investigar a viabilidade da aplicação da meta-heurística NCRO ao DSPSP baseado no domínio da SBSE. Para a escolha e discernimento do tema, duas questões foram definidas acerca do mesmo:

- *O que é e como é apresentado o tema*: o Problema do Escalonamento Dinâmico em Projeto de Software é um problema relacionado a associação de funcionários em tarefas de tal forma que a associação resulte em um menor custo, tempo de desenvolvimento, maior estabilidade e robustez diante das incertezas do mundo real. O problema é apresentado a partir de uma modelagem pertencente a SBSE, de tal forma que é possível trabalhar com metodologias multiobjetivas.
- *Qual a relevância para o estado atual da arte*: o fato de poucos trabalhos na literatura procurarem estudar o problema do escalonamento em sua forma dinâmica realça a importância desse estudo. Uma vez que, como mostrado em seções anteriores, um escalonamento pode ser crucial em um projeto de software, estudar e criar modelagens que incorporam a dinamicidade do mundo real acresce como uma fase importante em um pré-projeto de software.

A.2 Fontes de pesquisa

Todas as bases de dados acessadas para a construção desse TCC foram acessadas a partir do portal de periódicos CAPES[1]. As fontes selecionadas tiveram que obedecer os seguintes critérios:

1. Contém revistas atualizadas e artigos apresentados em congressos de Engenharia de Software.

[1] Disponível em: http://www.periodicos.capes.gov.br/

2. Possuem motores de buscas complexos, incrementando a pesquisa.
3. Possuem textos completos.

Assim sendo, foram usadas as bases de dados:

- *ACM Digital Library* (http://dl.acm.org)
- *IEEE Xplore* (http://ieeexplore.ieee.org)
- *Elsevier ScienceDirect* (http://www.sciencedirect.com)
- *SpringerLink* (http://www.springer.com)
- *SBSE Repository* (http://www.crestweb.cs.ucl.ac.uk/resources/sbse_repository)

Também foi usado como referência algumas referências usadas no trabalho de (SHEN et al., 2016), visto que este foi o modelo adotado para a investigação do Problema do Escalonamento Dinâmico em Projeto de Software. A literatura levantada foi organizada e mantida com suporte do software BibDesk, um software desenvolvido e mantido para macOS. Com o BibDesk foi possível editar e manter a bibliografia atualizada, pois seu ambiente permite guardar informações de acordo com sua natureza e ainda os organiza de forma categórica e ordenada. Assim, a ferramenta poupou desgaste manual na hora da escrita deste presente trabalho.

A.3 Palavras de buscas

A literatura encontrada é reflexo das palavras chaves utilizadas. Uma boa escolha de busca e uma complexa *string* para pesquisa incrementam a qualidade e permitem que os resultados sejam relacionados ao tema da pesquisa. Dessa forma, como já citado anteriormente, as bases de dados permitem ao usuário a possibilidade de dar como entrada uma *string* de busca, que se vale dos operadores booleanos *AND* e *OR*. A *string* usada para a pesquisa em todas as bases de dados foi:

((DYNAMIC OR (NOT DYNAMIC)) AND (SOFTWARE AND PROJECT AND SCHEDULING) AND (EVOLUTIONARY OR METAHEURISTIC OR OPTIMIZATION))

A exceção para a *string* foi a base *SBSE Repository*, pois esta não disponibiliza um mecanismo de busca. Dessa forma, o recurso utilizado para a procura de artigos relacionados ao tema foi feito a partir da ferramenta de busca do próprio navegador. Como consequência, todo o artigo cujo título continha referências a escalonamento foi coletado para a fase de seleção.

A.4 Seleção dos estudos

Mesmo tomando todos os procedimentos citados anteriormente da forma correta, ainda pode ser que certos textos selecionados previamente não se encaixem no tema ou não sejam relevantes ao mesmo. Essa motivação resultou na realização de passos para um refinamento sobre quais trabalhos realmente acresciam a realização deste trabalho. Os passos realizados foram:

1. Leitura do título do trabalho.
2. Leitura do resumo (*abstract*).
3. Leitura da conclusão.

Todos os trabalhos que passaram pelo critério acima e não foram descartados se submeteram a leitura completa, para uma última análise. A tabela 12 sintetiza o método de exclusão adotado a partir daqui.

Tabela 12 – Critérios de inclusão ou exclusão de um trabalho

Critério	Descrição
Inclusão	Trabalhos que apresentaram modelos de DSPSP.
	Trabalhos que levantaram estudos sobre o DSPSP.
	Trabalhos que investigaram o DSPSP no contexto da SBSE.
Exclusão	Trabalhos que não apresentaram resultados validados estatisticamente.
	Trabalhos curtos ou que fugissem do tema da engenharia de software.
	Trabalho repetido.

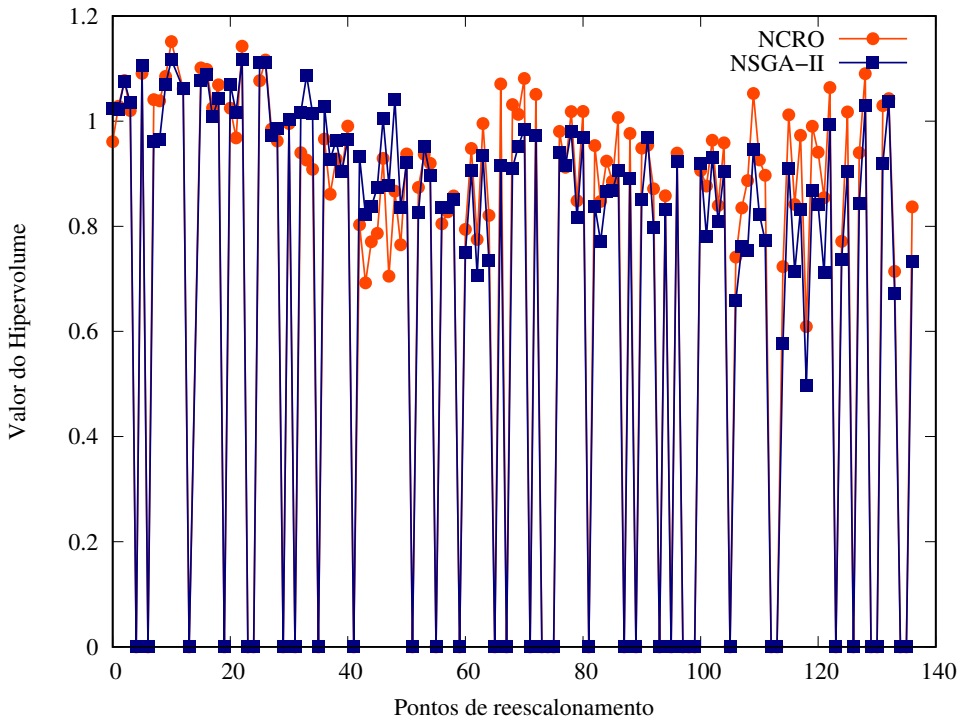
Fonte: Próprio autor.

De 15 trabalhos coletados da forma listada anteriormente, 5 foram considerados para a discussão nos trabalhos relacionados.

APÊNDICE B – Gráficos dos valores médios do hipervolumes em cada instância por evento

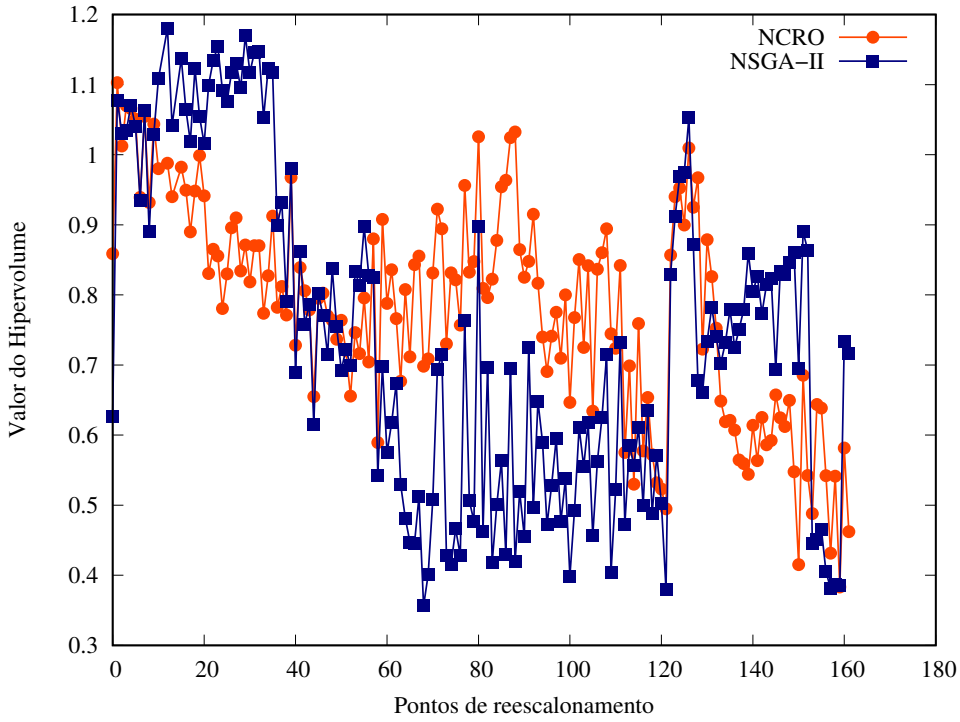
Este apêndice contém todos os gráficos gerados através da média dos hipervolumes de cada algoritmo, em cada evento.

Figura 30 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 1).



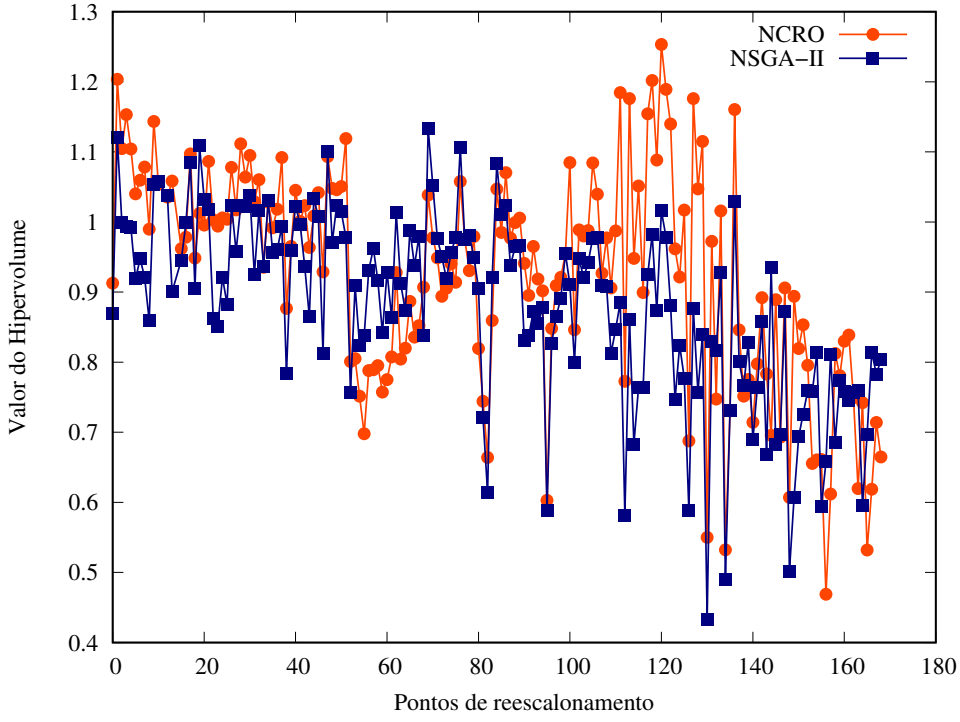
Fonte: Próprio autor.

Figura 31 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 2).



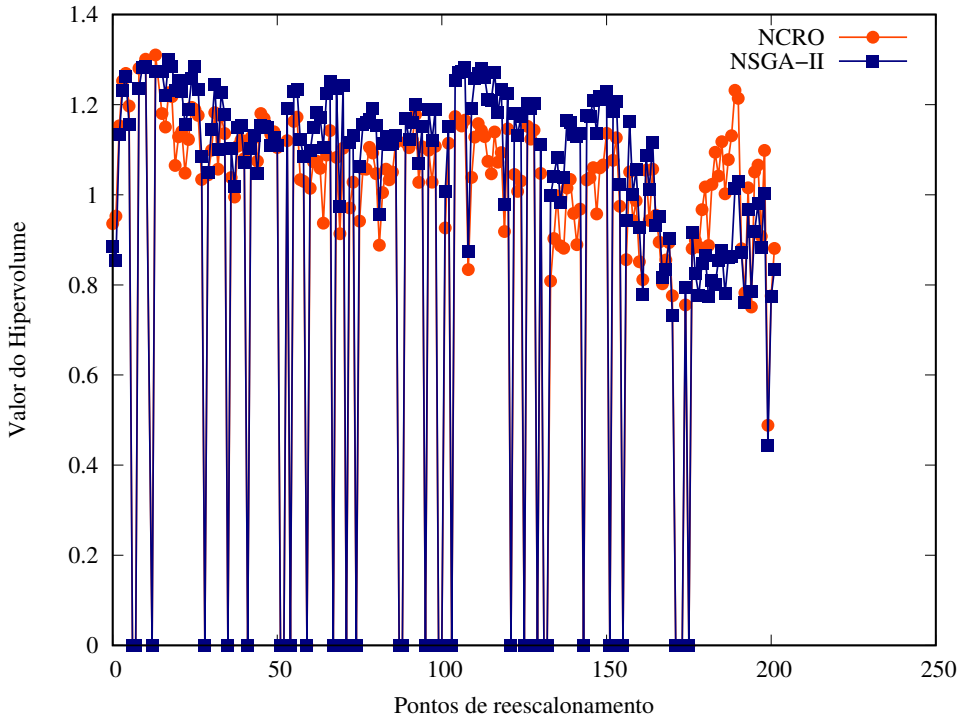
Fonte: Próprio autor.

Figura 32 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 3).



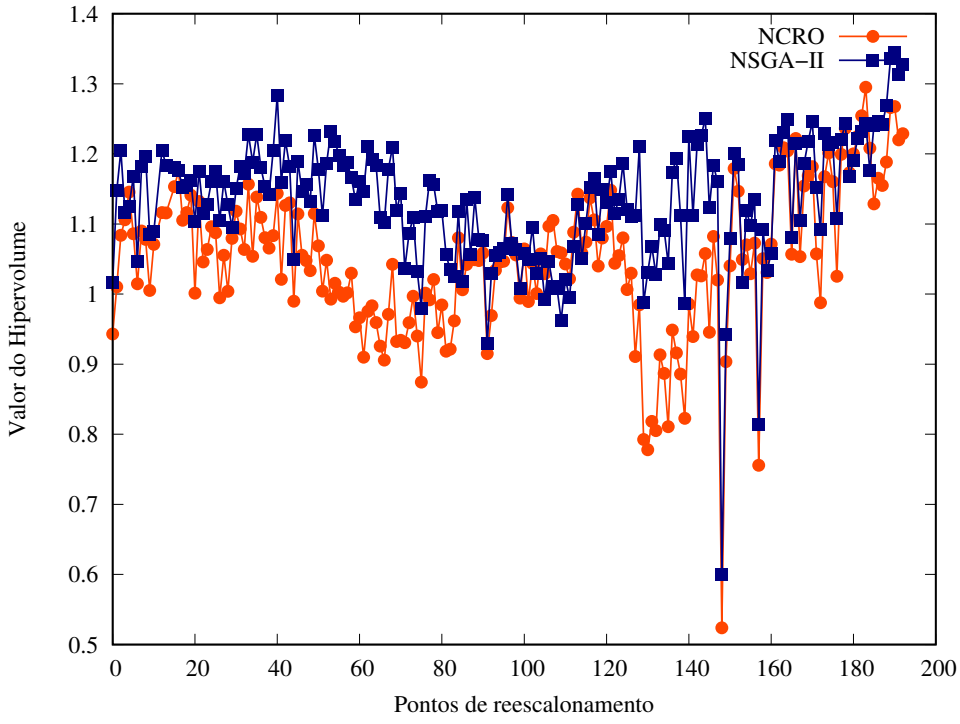
Fonte: Próprio autor.

Figura 33 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 4).



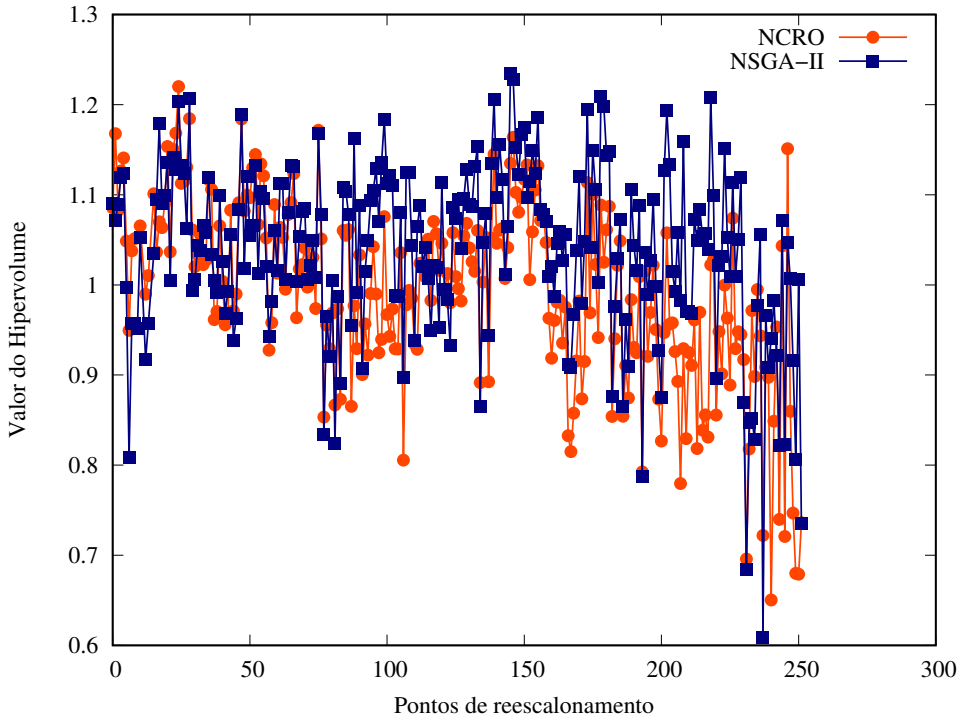
Fonte: Próprio autor.

Figura 34 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 5).



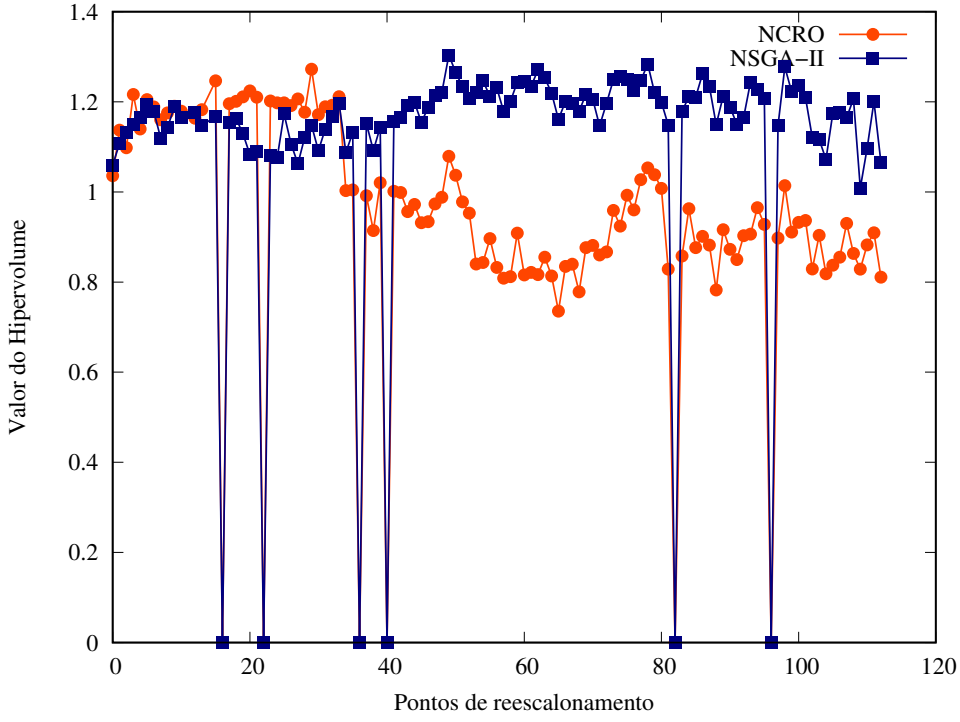
Fonte: Próprio autor.

Figura 35 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 6).



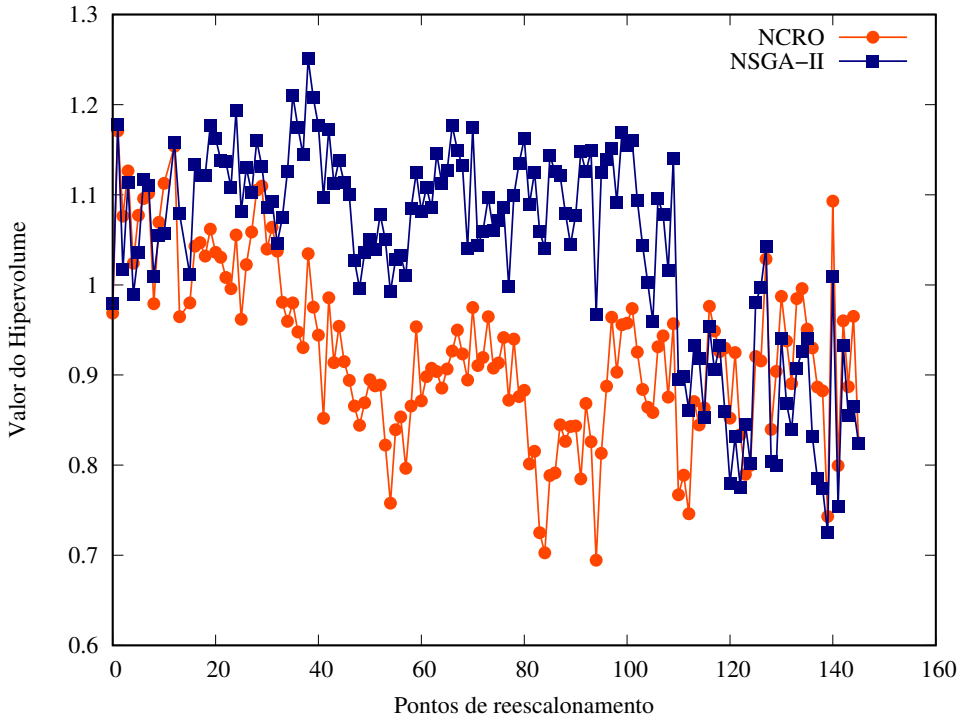
Fonte: Próprio autor.

Figura 36 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 7).



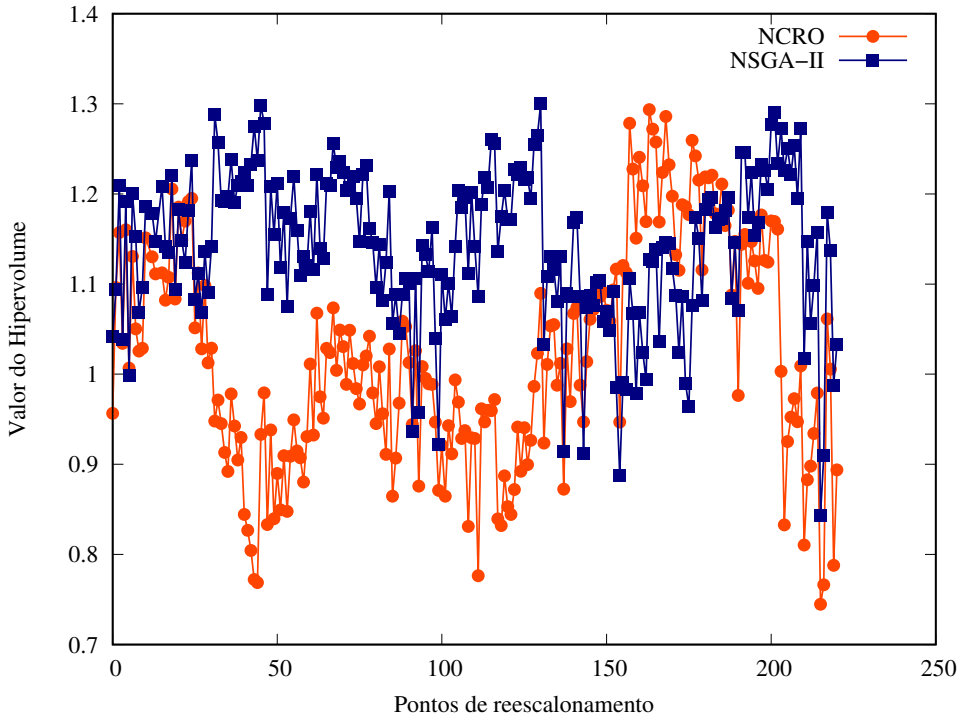
Fonte: Próprio autor.

Figura 37 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 8).



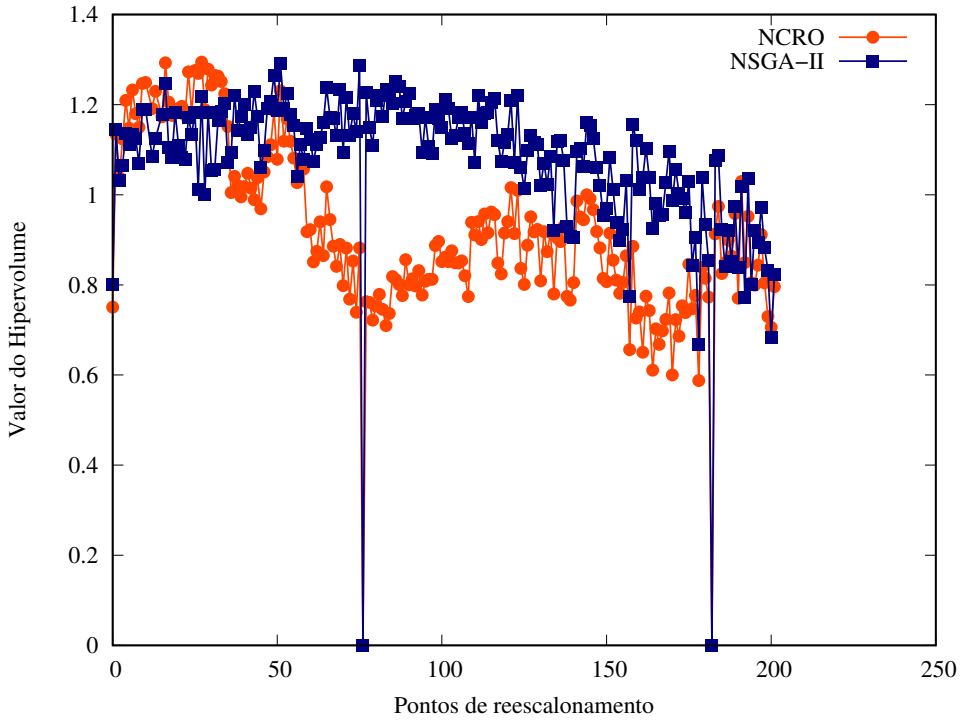
Fonte: Próprio autor.

Figura 38 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 9).



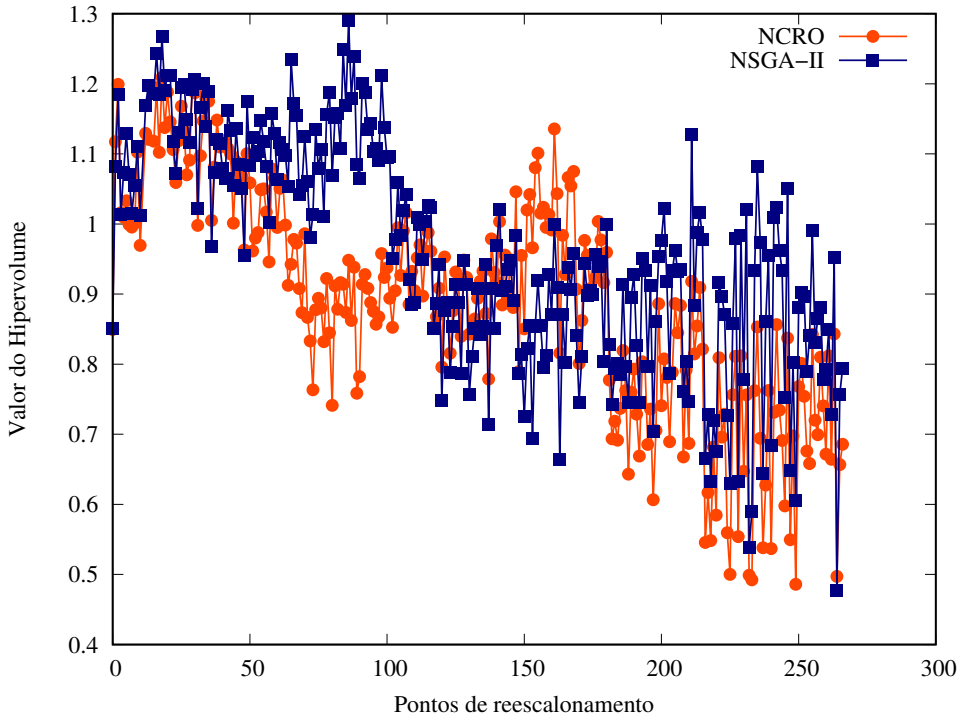
Fonte: Próprio autor.

Figura 39 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 10).



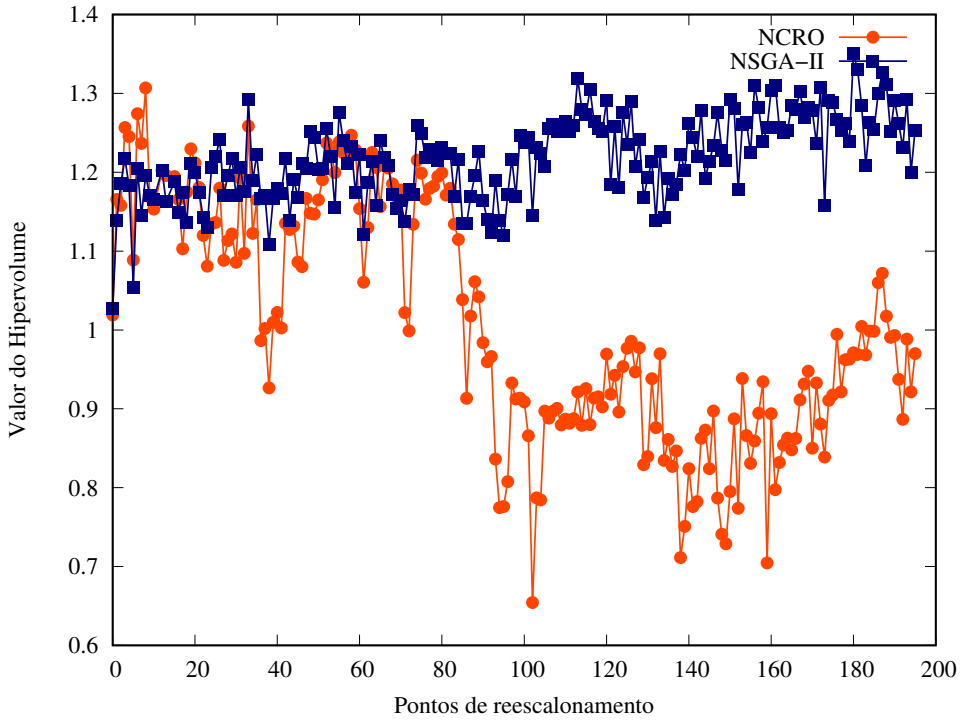
Fonte: Próprio autor.

Figura 40 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 11).



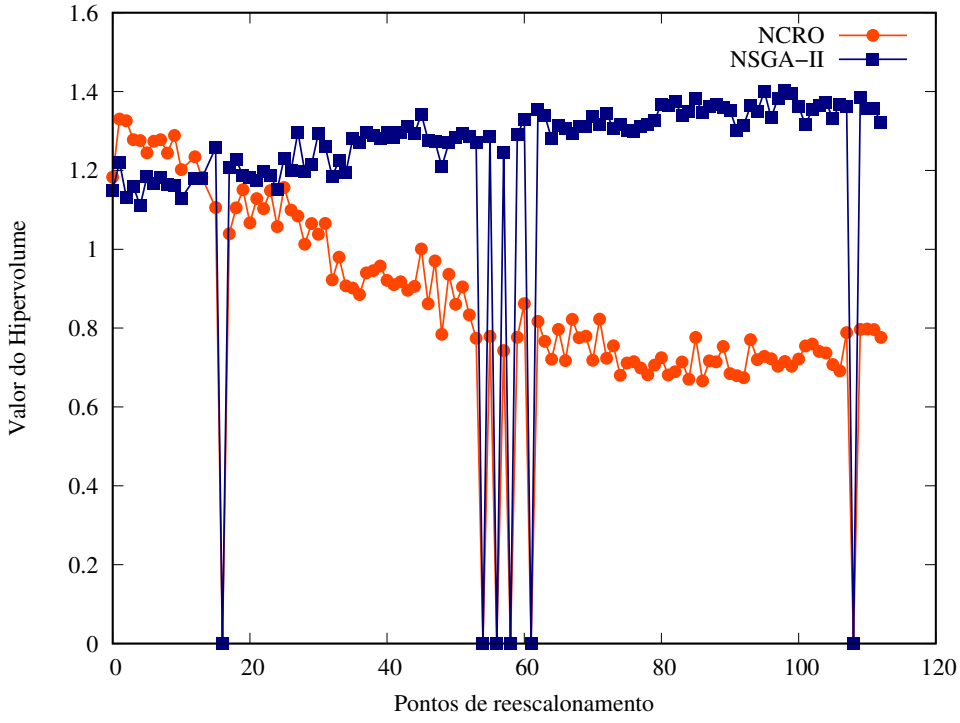
Fonte: Próprio autor.

Figura 41 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 12).



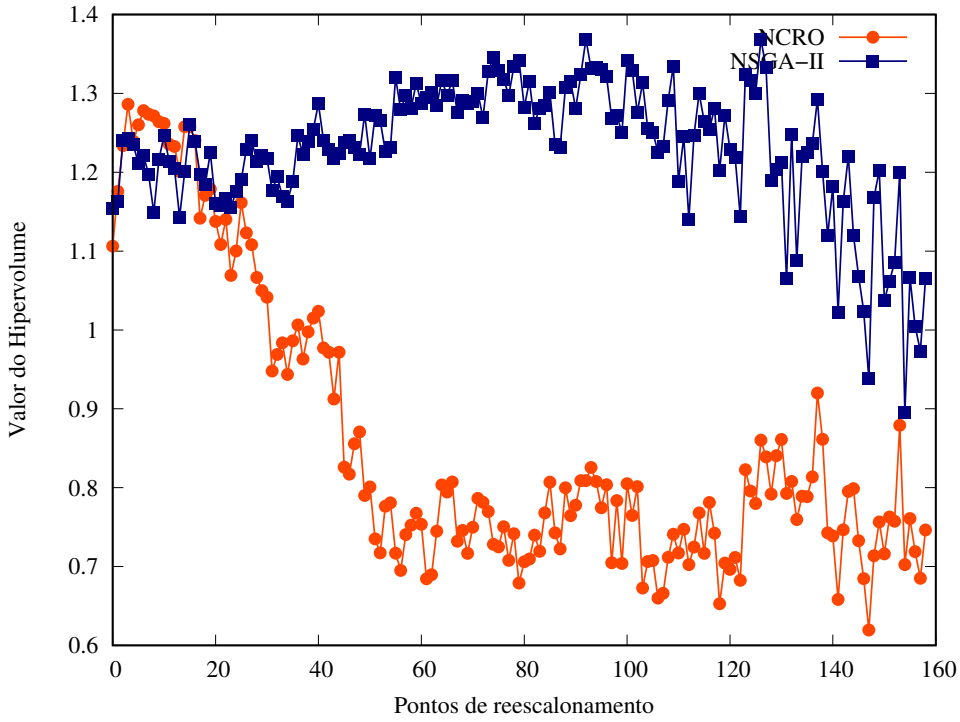
Fonte: Próprio autor.

Figura 42 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 13).



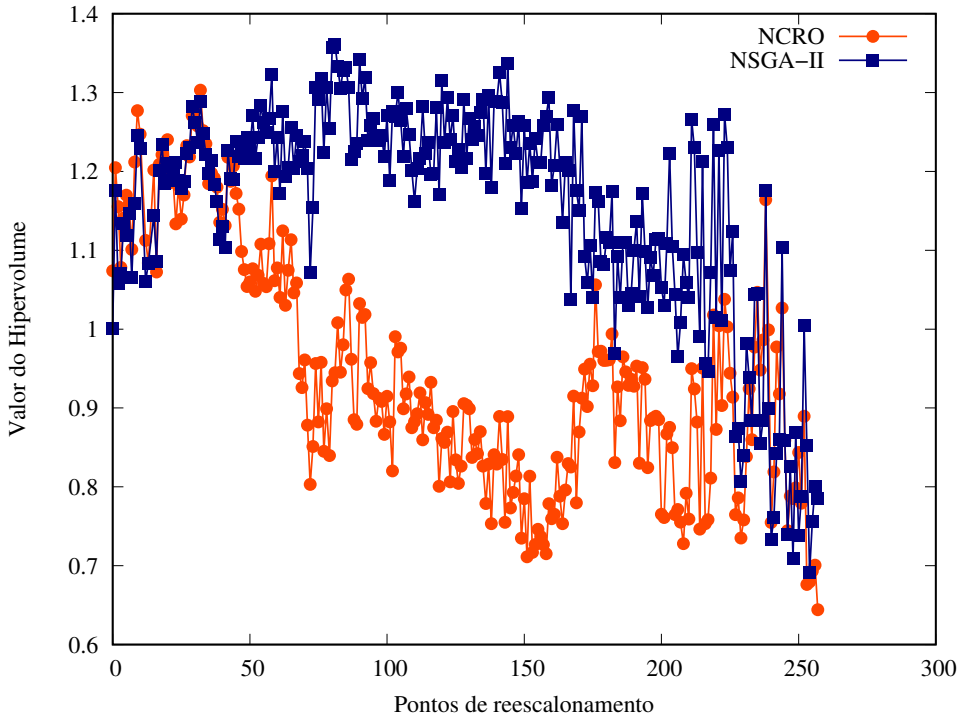
Fonte: Próprio autor.

Figura 43 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 14).



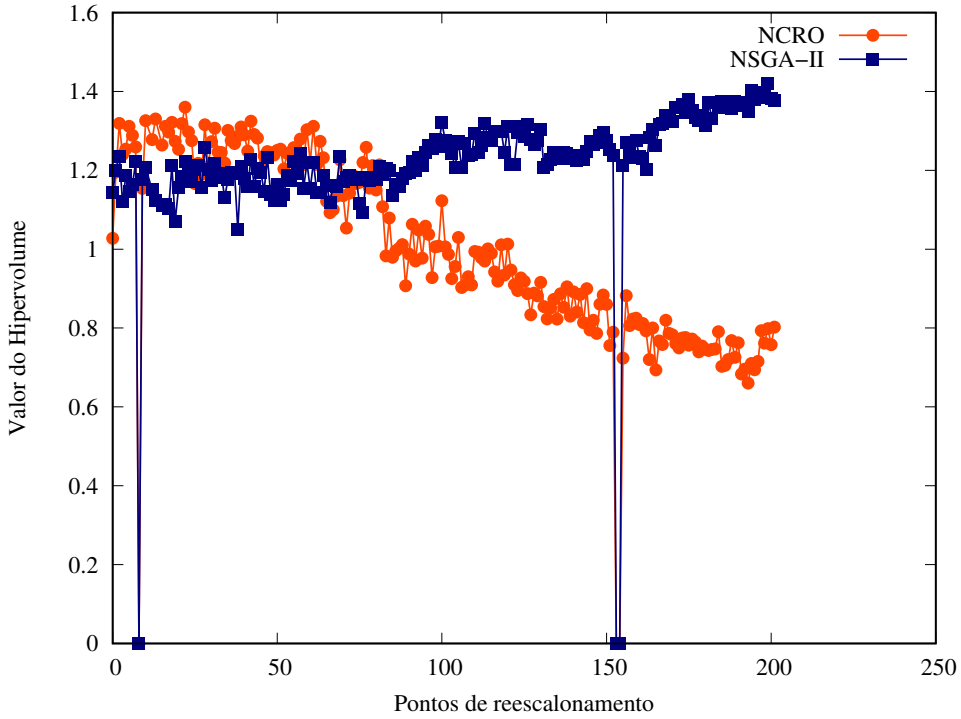
Fonte: Próprio autor.

Figura 44 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 15).



Fonte: Próprio autor.

Figura 45 – Gráfico dos hipervolumes à medida que o projeto era executado (instância 16).



Fonte: Próprio autor.